Exploiting the Shared Storage API

Alexandra Nisenoff
Carnegie Mellon University
Pittsburgh, PA, USA
nisenoff@cmu.edu

Deian Stefan
UC San Diego
La Jolla, CA, USA
deian@cs.ucsd.edu

Nicolas Christin
Carnegie Mellon University
Pittsburgh, PA, USA
nicolasc@cmu.edu

Abstract

As part of an effort to replace third-party cookies, Google introduced the Shared Storage API as one of their "Privacy Sandbox" proposals. The Shared Storage API seeks to replace some of the benign functionalities that third-party cookies facilitate while mitigating the potential privacy harms that they can cause, such as reidentifying users across websites. Shared Storage seeks to do this by allowing third parties to store data that is not partitioned by top-level website, but limiting read access to those data.

We find that the implementation and design of the API have flaws that allow for both the reidentification of users across sites and the leakage of more data than intended by Google. With the API being deployed in Google Chrome and major advertisers and trackers having completed the processes required to gain access to the API, the Shared Storage API may not do as much as intended to improve the state of privacy on the web. We present several attacks on the API that circumvent the key goals laid out by Google as well as discuss potential extensions and mitigation strategies. While we have responsibly disclosed our attacks to Google, most attacks remain possible in Chrome.

CCS Concepts

• **Security and privacy**; • **Information systems** → **Online advertising**; *Browsers*;

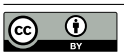
Keywords

Web Privacy; Privacy Sandbox; Online Advertising; Online Tracking; Privacy

ACM Reference Format:
Alexandra Nisenoff, Deian Stefan, and Nicolas Christin. 2025. Exploiting the Shared Storage API. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744848

1 Introduction

As users go about their daily online lives, they are being tracked by companies seeking to monetize information about them. This tracking can take on many forms, from HTTP cookies to browser fingerprinting [21], and is conducted by both first parties (i.e., the domain that a user directly visits) and third parties (i.e., all domains other than the one that the user directly visits).



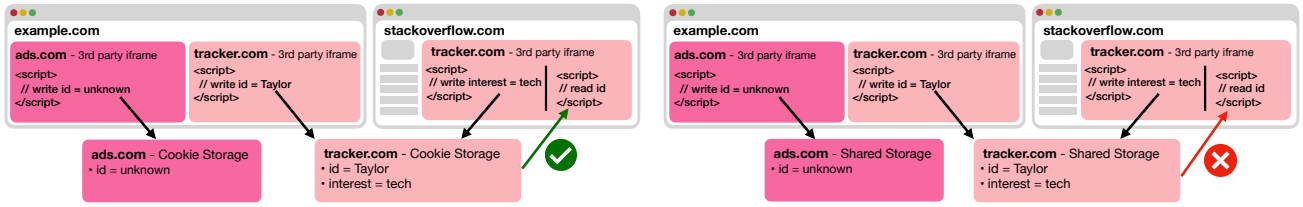
This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '25, Taipei, Taiwan
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3744848

Tracking can violate user privacy, direct users to more expensive products [64], and discriminate against users even based on legally protected attributes [5, 17, 39, 70, 83]. Online tracking is a fundamental part of the multi-billion dollar digital advertisement industry, and allows companies to build up information about users' demographics, interests, and activities. Advertisers then use this information to show individual users ads they believe will be more relevant and effective [57, 79, 80]. Further, user information is frequently shared and sold [77] in a way that is opaque to users.

In response to this tracking and the negative effects that it can have on end users, many browsers have taken steps to protect users, from partitioning storage to entirely deprecating third-party cookies [9, 59, 82]. For several years Google had plans to deprecate third-party cookies while introducing several APIs (primarily related to advertising), under the umbrella of their "Privacy Sandbox." After many delays in their third-party cookie deprecation timeline [28], Google finally stated that it plans to leave third-party cookies enabled in Chrome while also continuing to develop the Privacy Sandbox APIs [4].

In this paper, we discuss one component of Google's Privacy Sandbox—the *Shared Storage API*. Google created the Shared Storage API to be a low-level, multipurpose, and privacy preserving replacement for third-party cookies. At the highest level, the API provides per-origin memory that is not partitioned by top level site, the same way third-party cookies are currently handled in Chrome, but with limited read access to the data. Google specifically says that the Shared Storage API "seeks to avoid the privacy loss and abuses that third-party cookies have enabled. In particular, it aims to limit cross-site re-identification of user(s)" [76]. While marketed as a multipurpose API, many of the use cases that Google suggests are directly related to advertising. This includes use cases such as limiting how many times a user has seen a specific ad or reporting how many unique users have seen an ad [51].

In this paper, we analyze the Shared Storage API to discover the extent to which the current implementation and underlying proposal meet its privacy goals. We discover and describe several attacks that circumvent the protections that the API is supposed to offer. Some of the issues we uncovered include several covert timing channels, a caching covert channel, and a major limitation of how the proposal quantifies the amount of data that can be leaked with a single Shared Storage API operation. Several of these issues are similar to the limitations of other Privacy Sandbox proposals. We further discuss extensions to these attacks and possible ways to mitigate them: though eliminating them altogether seems unlikely without severe degradation to the usability of the API. Indeed, we have disclosed these vulnerabilities to Google, and several of the attacks remain feasible at the time of writing. To the best of our knowledge, the only publicly disclosed issue with the Shared Storage API prior to our work was about how much data can be



(a) Third-Party Cookies **(b) Shared Storage**

Figure 1: Partitioning of cookies and shared storage data in Chrome

leaked over time through standard use of Shared Storage. We discuss this problem for completeness and compare it to a similar issue with federated learning of cohorts (FLoC) [30].

Creating APIs and tools that properly protect user privacy is an incredibly important yet difficult challenge. The Shared Storage API does have more limitations on data access than traditional third-party cookies, but it currently fails to live up to its stated goals and introduces new challenges. With Chrome being the most widely used web browser, with over 60% of the browser market share [25], any new feature that is rolled out in Chrome has the potential to impact a large number of users. Despite concerns about the API having been raised by other browser vendors in the normal course of W3C discussion (see Section 6.4), the Shared Storage API is currently available in Chrome and has been since as early as June 2022 in the canary and developer versions of Chrome [76]. This paper also provides a first look at the use of the API in the wild, despite privacy issues inherent to the design and implementation.

2 Background

Next, we describe relevant background, including how cookies can be used for tracking and the relevant details about how the Shared Storage API works.

Third-Party Cookies: Chrome currently allows arbitrary third parties to utilize cookies. As Figure 1a shows, the third parties can both write to and read from the same shared cookie storage across top-level sites. If a user first visits `example.com` which includes content from `tracker.com`, `tracker.com` can set a cookie that represents the user's identity. For the sake of this example, we will say that the identifier for the user is "Taylor." `tracker.com` can then record that Taylor has visited `example.com`.

When Taylor visits `stackoverflow.com` that also includes content from `tracker.com`, `tracker.com` can read that identifier that has been stored as a cookie and record that they have visited `stackoverflow.com`. They can also choose to make the assumption that Taylor is interested in technology. Since `tracker.com` has already seen that identifier before they can start to build up a profile about what Taylor is interested in based on the sites they have visited and any other auxiliary information they might have. While being interested in technology may be fairly innocuous, websites could reveal something potentially more sensitive such as health conditions, sexual orientation, or political affiliation. The issue here is that by being able to read the cookie value the user can be re-identified across websites by a third party on multiple sites visited by the same user.

Shared Storage: This is where the Shared Storage API comes into play. As Figure 1b shows, while writing data to storage that is unpartitiond by top-level site is still allowed, the reading of values through the Shared Storage API is intended to be severely limited. While the cookies are sent alongside network requests and can sometimes be read by JavaScript, values in shared storage cannot be directly read. The idea is that even if the website were to store "Taylor" in shared storage if that value cannot be read, then the user cannot be re-identified.

Shared Storage attempts to accomplish this by only allowing data stored in shared storage to be read in an isolated environment, called a *worklet*.

One main use of the Shared Storage API is deciding what content to show a user based on values stored in shared storage. This can be done by calling the Select URL API which takes in up to eight URLs and selects one to be shown to the user based on the values in shared storage. The Select URL API can not be used without the Shared Storage and consists of a single function: `selectURL`.

Figure 2 shows this `selectURL` being used for A/B testing by `tracker.com`. With cookies this would be very simple, one would simply read the cookie through the JavaScript of the webpage and use an `if` statement to set the `src` attribute of an iframe to the URL of the content the user should see.

Since the values in shared storage cannot be directly read outside of the worklet, the branching logic must happen inside the worklet. A call to `selectURL` triggers code inside the worklet. From there, the code can decide which URL to show the user based on the shared storage data and any auxiliary information passed in with the `selectURL` function call. When the selectURL function returns the content, the selected URL can be loaded either as an iframe or a fenced frame.

Non Worklet Functions: In addition to functions that cause code to run in the worklet. Developers can call Shared Storage functions such as `set`[1], `append`, `batchUpdate`, `delete`, and `clear` to modify values in shared storage from the JavaScript of a page or in network response headers in much the same way that cookies can be set.

Worklet Functions: The worklet has access to all of the Shared Storage functions that are available to the JavaScript outside of the worklet but with additional functionality such as being able to use Private Aggregation API operations.

[1]To prevent overwriting existing data the `ignoreIfPresent` attribute can be used while calling the `set` function.

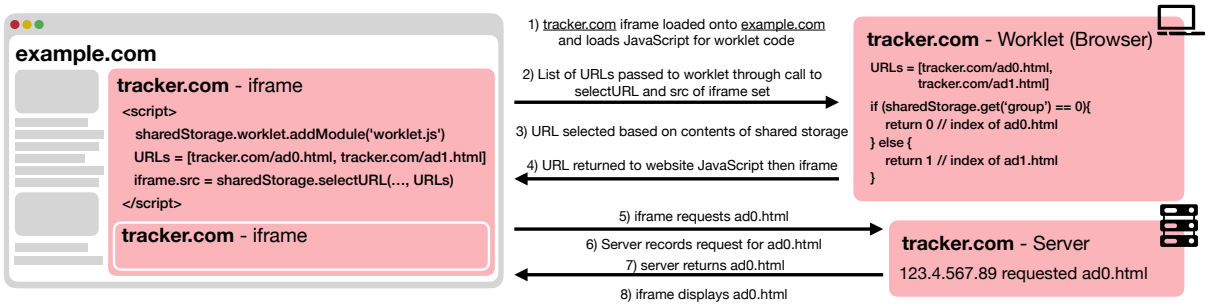


Figure 2: Selecting a URL from a list using the Shared Storage API for a user where the value for group has been set to 0.

Code that runs in the worklet must be defined in a separate JavaScript file and loaded into the page prior to being invoked using either the `addModule` or `createWorklet` functions.

Select URL Budget: While the number of URLs passed into `selectURL` limits the amount of information that can be learned about the values with a single call, *each* time `selectURL` is called some information could be leaked. This means that an unlimited number of calls could leak an unlimited amount of information. To limit how much information is leaked over short periods of time the API introduces several "budgets." With a starting budget of 12 bits per origin, across all pages visited by a user in a day, each call to `selectURL` costs $log_2(|URLs|)$. The reasoning is that the $log_2(|URLs|)$ deduction represents the information theoretic number of bits that are leaked by knowing which of $|URLs|$ was selected. If only two URLs are passed in, there are only two possible outputs (i.e., URLs that could be selected), which could be represented by a single bit (0 or 1). Similarly, if eight URLs are passed in there would be a budget deduction of three because it would take three bits to represent eight values. If the deduction for a given operation would make the total of the budget deductions in the previous day exceed the budget, the call would return the first element in the list of URLs that have been passed in.

Google has also added two additional budgets to the API as a way to patch issues in the proposal. To prevent a visit to any one page from leaking too much there is an additional budget of 12 bits per top-level page load. The other new budget seeks to prevent a single origin from monopolizing this per page page load budget giving each an individual budget of 6 bits per top-level page load.

Data Origin: For the most part, the origin of the context used to invoke Shared Storage API functions determines the origin of the data that the functions can access. However, cross-origin worklets can be created allowing third parties to access their data when using functions such as `selectURL` without the need for an iframe.

Operation Queue: Calls to all Shared Storage API functions are placed in a queue for the corresponding origin responsible for the function call and are executed in the order that they were added to the queue. Having a unique queue for each origin prevents the API from leaking some information about an origin's actions to other origins on the page.

Having this queue also means that calls to operations that may take time to finish executing are completed before subsequent functions execute. For example, if two calls to `selectURL` are made, the first must finish executing before the second starts running. If there were no queue, developers would not be able to reason about whether a call to functions like `set` or `append` would execute before a call to `selectURL` returned.

Data Persistence: By default, the values stored in shared storage are retained for the 30 days following when they were written. This time limit is far shorter than the current 400-day maximum lifetime of a cookie in Chrome [14]. Users can also manually clear shared storage values, along with other browser data, including cookies, from the Chrome settings menu.

Attestation Requirement: A non-technical measure that Google put in place to prevent misuse of the APIs is the requirement that companies complete an enrollment and attestation process before they can use the privacy sandbox APIs (including the Shared Storage API) on websites. During the process of requesting access to the APIs companies must explicitly state that they do not intend to identify users across sites. Companies can do this by filling out a form from Google, where they must also specify which APIs they want to use, as well as information about their company such as the location of their privacy policy.

Once they receive approval, which is tied to a domain, they are given a file that they must place at a standardized `well-known` URL. Users may also manually list out domains that do not have to complete the attestation process to access the privacy sandbox APIs on their browser. This requirement went into place in August 2023 [35] and the implications of this requirement are discussed in more depth in Section 6.

Fenced Frame: As mentioned earlier, the result of `selectURL` can be loaded into a fenced frame. Fenced frames are another one of Google's privacy sandbox proposals [31]. Functionally, they perform a similar function to iframes creating a nested browsing context in a page. The main difference is that communication between the content of the fenced frame and the embedding context is highly restricted. Eventually Google will require that the result of `selectURL` be loaded into a fenced frame but, Google is allowing the URLs to be loaded into iframes through 2026 [76].

Private Aggregation API: The Shared Storage API can also be used in conjunction with Google's Private Aggregation API. As the name suggests the API can be used to create reports of aggregated data, including data from shared storage [32]. Use cases for this include reporting on approximately how many unique users have seen an ad or the demographics of those users. To preserve privacy,

these reports do not include exact numbers, but also include some level of noise.

The Shared Storage proposal also currently allows event level reporting which allows developers to send reports triggered by events in the fenced frame to URLs specified in the initial call to `selectURL`. Like fenced frames, this less privacy preserving temporary measure will be allowed through 2026 [76].

3 Threat Model & Attacker Capabilities

We evaluate the feasibility of attacks from a third-party attacker against Shared Storage. The attacker has previously stored data (e.g., a unique identifier) in shared storage, in any context, and aims to circumvent the Shared Storage budget and read those data when the user visits a different page. We primarily discuss how those data could be used by third parties to re-identify users, because that is precisely what the Shared Storage API intends to prevent, but those data could be used for other purposes as well.

For an attacker to use the Shared Storage API at all, and subsequently for our attacks to work, they must have completed the attestation process as described in Section 2. This would be true for developers intending to use the Shared Storage API for legitimate reasons. The same process would need to be completed by companies if they wanted to make use of the attacks that have been shown to work against other Privacy Sandbox APIs such as the Topics API or the Protected Audience API.

Rather than considering an attacker that can directly load any content (e.g., an image or JavaScript file) onto the top level page that is not controlled by them, we assume that the attacker is *only* able to load their content into an iframe on the page.[2] The attacker model and capabilities of the attacker described above are meant to mimic the situation where content from an advertiser or known tracker is loaded into an iframe.

4 Attacks

In this section we present several methods for the attacker described in Section 3 to circumvent the protections that the Shared Storage API is designed to provide. Specifically, we identify a network timing attack, a crash attack, a queue timing attack, and a caching attack. We discuss these next. We also discuss the known issue of the API's budget resets leaking information over time. A summary of Google's responses to disclosures of the attacks described in this paper can be found in Section 4.6.

4.1 Network Timing Attack

By definition, a network request to a URL selected by `selectURL` cannot be made before the function in the worklet returns (i.e., in Figure 2, step 5 must happen after step 4). Through the attack described below, this allows us to exfiltrate data from shared storage. As the function called by `selectURL` runs in a worklet, it has unfettered access to the values stored in shared storage. This means that the function's behavior, including how long it takes for the function to return, can vary based on these stored values.

[2]Google recently added support for cross-origin worklets as described in Section 2. However, we describe the attacks in the rest of the paper in the context of an attacker that simply has content loaded into an iframe.

Writing a Value: Writing a value that will subsequently be used for tracking can be done in the standard way any data can be written to shared storage (e.g., with JavaScript code or via response headers). The pseudocode in Algorithm 1 gives an example of how this can be done by an attacker iframe (running JavaScript) for a single binary variable stored in shared storage.

Algorithm 1 Network Timing Attack - Write

```
1: procedure WRITE(identifier)
2:     counter ← 0
3:     for bit in identifier do
4:         window.sharedStorage.set("counter",
               bit, {ignoreIfPresent:true});
5:         counter ← counter + 1
6:     end for
7: end procedure
```

Algorithm 2 Network Timing Attack - Read

```
1: procedure WORKLET(counter)
2:     value ←sharedStorage.get(counter)
3:     if value == 1 then
4:         Sleep for a predetermined amount of time
5:     end if
6:     return 0                    ▷ Network request for URL made
7: end procedure
   procedure PAGE JS(worklet_script, URL_list)
2:     sharedStorage.worklet.addModule(worklet_script);
       counter ← 0
4:     while counter < length of identifier do
           make a nested iframe i                  ▷ or fenced frames
6:         opaqueURL ← await sharedStorage.selectURL(
                       counter, URL_list);
           i.src ← opaqueURL;
8:         counter ← counter + 1
       end while
10: end procedure
```

Reading a Value: Algorithm 2 shows how to read this value. Within an iframe they control, the attacker creates as many (nested) iframes as there are bits in their identifier, to load the results of `selectURL` and make network requests. From there, the attacker can call `selectURL` as previously described and immediately set the `src` attribute of the iframe or the equivalent attribute for a fenced frame to the value that the call to `selectURL` returns. To prevent the code that called `selectURL` from learning when the function in the worklet returns, a promise is returned as soon as the call is placed in the operation queue which resolves when the function in the worklet returns and the embedding page is prevented from directly reading the value of the URL that was returned. Inside the worklet, if the function sees that the first bit of the data[3] it wants to exfiltrate out of the worklet is a 0, it returns immediately; if it sees a 1, it waits before returning. When the function in the worklet

[3]For reidentification of a user across sites this data would be an identifier.

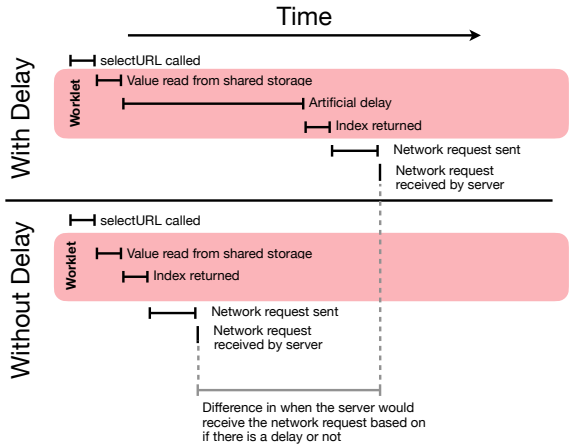


Figure 3: Server-side timing difference with delayed worklet.

returns the promise from `selectURL` resolves, a network request can be made, and the server can learn what value was stored in the first bit of the data based on the relative time that it receives a request relative to some earlier request or through auxiliary information sent from the browser about when the call to `selectURL` was made. Figure 3 shows the difference in the time at which the server receives a request generated by a call to `selectURL`, based on how long the function in the worklet takes to return (i.e., if an attacker chooses to add an artificial delay to the function in the worklet).

By doing this, an attacker can leak an additional bit of information that will not be accounted for in the existing privacy budgets. This is because the proposal only considers information leakage based on which URL is selected:[4] not any covert channels. Hence, how much data can be transmitted and how quickly depends on how long the user stays on the page and the delay parameter used by the attacker.

Practicality: To understand if this attack is practical, we must consider the average time a user spends on a page and the length of an identifier that an attacker wants to leak. For data to leak with this attack, the user must remain on the page through when the call to `selectURL` returns so that the page can make a network request to the server, leaking the information. Since calls to `selectURL` happen in series rather than in parallel, if an attacker wanted to leak two bits of data, each one requiring a delay, the user would need to stay on the page for a minimum of approximately 2 delays.

Prior works have estimated that the median time that a user stays on a webpage ranges from around 12 seconds to just under a minute [36, 47, 52, 53]. We also take the identifier to be 33 bits long, as that is the minimum number of bits required to uniquely identify every person on Earth ($2^{33} \approx 8.5$ Bn).

To determine the longest delay that would allow a 33-bit long identifier, that requires 33 delays, to be leaked within the median time a user spends on a page, we can simply divide the number of seconds by 33. A longer delay may be more noticeable to the server as it is less likely to be due to network conditions. Since the iframes that these resources are loaded into can be hidden from users, even a longer delay could go unnoticed by users. By doing this, we find that the maximum delay would be between 0.36 seconds (for a 12s dwell time) and 1.82 seconds (for a 1-min dwell time). For identifiers that do not require a delay to transmit every bit, the maximum delay could be larger since finishing all of the calls to `selectURL` would happen more quickly. A delay of 0.36 seconds is in line with the round-trip time between the US and Australia, meaning that an additional delay of that length would likely allow an attacker to distinguish an intentional delay from normal variations in the latency of the request.

Even in the case where a full identifier cannot be transmitted, the information that *is* leaked can be used in conjunction with other methods of identifying users (e.g., browser fingerprinting) to improve a tracker's ability to re-identify a user if the Shared Storage API had not existed.

Extensions: Several slight modifications can extend this attack by making it faster, avoiding any deduction from the Select URL budget, or leaking more information per call to `selectURL`.

Avoiding Select URL budget deduction: If JavaScript makes a call to `selectURL` and only passes in a single URL the call will not decrease the Select URL budget ($\log_2(1) = 0$). Paired with this covert channel, this means that information about the values in shared storage can be leaked without any deduction from the Select URL budget as long as the user remains on the same page.

Leaking more than one bit with a single call to `selectURL`: We assumed above that this covert channel uses a single predetermined delay, but, conceptually, there is no reason an attacker could not use different delays to pass more information to the server (e.g., no delay, short delay, long delay to reveal which of three values were stored). This may be more beneficial in a scenario in which limiting the number of requests being made is more of a priority than the amount of time it takes to convey information.

Optimizing the speed at which data can be leaked: Tailoring delays to current network conditions would be useful to balance how long it takes to reveal the information, while ensuring that the delay is long enough that it is not obfuscated by normal variability in network conditions. An attacker could simply time how long the request for the worklet script from `addModule` took to get a sense of the current network conditions without requiring an unnecessary network request. Repeating the transmission of the value could also help with any noise introduced by network variability.

If a tracker were creating identifiers, they could optimize their selection to assign identifiers that minimize the number of delays used to transmit that information, thereby decreasing the total time needed to convey information to the server. Furthermore, putting as many of these values towards the beginning of the identifier would allow the largest number of bits to be sent in the shortest amount of time in case the user left the page before the entire identifier could be transmitted.

Potential Mitigations: As an attacker can currently add arbitrary delays to functions in the worklet, attempting to mitigate this covert channel by adding small delays to the return times of the code in the worklet could easily be defeated by the attacker making their delays longer. Concretely limiting the runtime of functions in the

[4] In this scenario, the actual URL that is returned by the call to `selectURL` can also be used to reveal information concurrently with the delay as long as the select URL budget has not been exhausted.

worklet could also help with this issue. In a public post, made after our disclosure, the developers did mention potentially adding a limit to the run time of another worklet function [45], suggesting their openness to limiting function run times in the worklet.

In their response to our disclosure, Google mentioned various ad-hoc solutions from requiring "that all of the URLs [be] prefetched as web bundles," to "fetch[ing] from some CDN that has a trusted policy that it won't leak its logs," to requiring "the server [execute] in a trusted execution environment running trusted code", and using "some sort of private information retrieval service." As URLs may point to dynamic content or additional external resources, both the initial request and *any* subsequent requests would need to be thus restricted. These solutions are not just ad hoc, but also heavy-weight and a significant departure from open web standards.

Adding these types of network access restrictions to fenced frames has also been discussed [67]. But these plans have not been implemented and, as long as the result of `selectURL` can be loaded into an iframe, the benefits of fenced frames are a moot point.

By preventing calls to `selectURL` with one URL the API could limit how much information is leaked through several attacks described in this paper. When calling `selectURL` with a single URL only one URL can be returned and could simply be loaded directly.[5]

4.2 Crash Attack

When the network conditions (e.g., slow mobile networks) require prohibitively long, or highly variable delays, using a *termination* covert channel (instead of an external timing covert channel) may make it easier for an attacker to leak data.

Writing a Value: Writing an identifier is done in the same way as the previous attack: the attacker simply writes values to shared storage using JavaScript or via response headers.

Reading a Value: To read a value, the attacker goes through the same steps as in the previous attack, but instead of delaying when the function returns, the attacker intentionally crashes the JavaScript in the worklet depending on the value stored. When the worklet JavaScript crashes, no request is made to the server. Like in the previous attack, the server can then infer the value stored in shared storage depending on whether it receives a request (i.e., the worklet has not crashed), or not (i.e., the worklet has crashed). The deductions from the Select URL budget take place whenever the navigation of a frame occurs since that is when the information is hypothetically leaked to the server. Crashing the worklet prevents this navigation, and therefore, budget deduction from happening.

In terms of implementation, we initially found a bug that allowed us to crash the worklet by calling `console.log` on an array inside the worklet. While this specific bug was patched, a simple strategy involving computations on a massive array could still crash the worklet, and prevent a request from being made.

Besides the worklet, other elements on the page will also crash in accordance with Chrome's site isolation policy [63]. For instance, an iframe embedding a worklet that crashes itself also crashes, preventing subsequent requests. In Figure 4, this means that everything from `tracker2.com` will crash. To handle this, the attacker

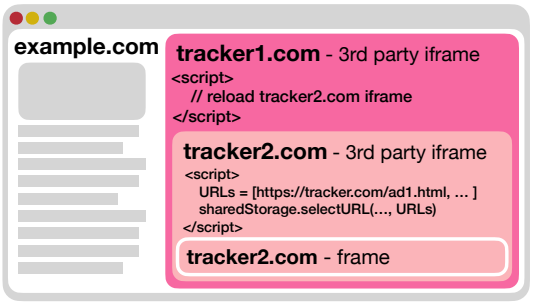


Figure 4: The iframe format necessary for the crash attack.

simply needs to nest the `tracker2.com` iframes in another iframe from a separate domain (that does not need to have completed the attestation process). In the figure, this is `tracker1.com`. Since `tracker1.com` is from a different domain it will not crash with `tracker2.com`. From here, the `tracker1.com` iframe can reset the `src` attribute of the top most `tracker2.com` iframe forcing the frame to reload. By reloading the nested iframe, the attacker is given another opportunity to either make a request or crash, leaking another bit in the same way the first bit was leaked.

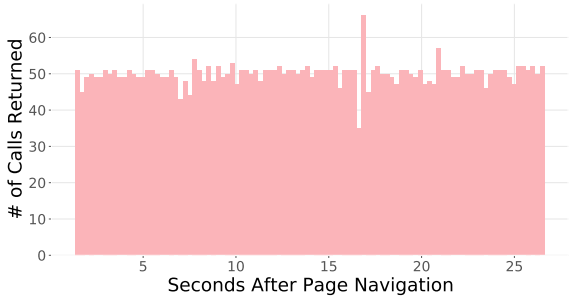
Practicality: To assess the practicality of this attack, we implement it as described above and experimentally determine what delay between reloading the outermost iframe is necessary to consistently allow the worklet to crash or make a request for a resource. Our tests were completed on a 2020 MacBook Pro with 32 GB of memory and a 2.3 GHz Quad-Core Intel Core i7 processor running Chrome version 135.0.7049.115. We find that with a delay of approximately 0.5 seconds, we can consistently recover randomly selected identifiers. With delays shorter than this we begin to see a degradation in performance: not all bits of the identifier are exfiltrated. In these cases, partial data are still revealed, which could be used in conjunction with auxiliary information to aid in reidentifying a user. If we consider an identifier of 33 bits this would require a user to stay on a page for 16.5 seconds. While this delay falls within the range of median dwell times identified in the literature, it is potentially slower than the network timing attack. We stress that this is simply a ballpark estimation, as it is likely that the necessary delay between reloads of the page is deeply dependent on several other factors including, but not limited to, the hardware used to run Chrome and the method used to crash the worklet.

Extensions: As with the previous attack, calling `selectURL` with a single URL would result in no budget deduction, even if the JavaScript did not crash. Crashing the worklet can also be paired with delaying requests to leak more information.

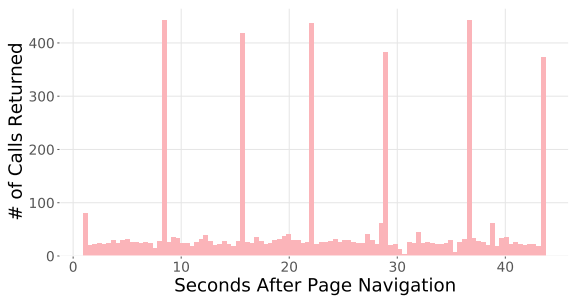
A similar result can be achieved by causing a crash in the fenced frame that is loaded as a result of a `selectURL` call. Similar to crashing a worklet, crashing a fenced frame causes cascading failures. This could be used to circumvent mitigation strategies that depend on pre-loading all possible resources as it leaks information to the embedding frame.

Potential Mitigations: If a worklet crashes, preventing the frame that calls it from crashing and returning the default URL could prevent the server from learning about the crash, since the server would still receive a request. Unfortunately, developers might not

[5] All other worklet operations that otherwise would have been run when calling `selectURL` could be handled with the `run` function which allows code to run in the worklet without selecting a URL.



(a) Only webpage JavaScript writing to shared storage.



(b) Webpage JavaScript and worklet writing to shared storage.

Figure 5: Return times of calls to set from webpage JavaScript

expect this behavior from the API and this would also (erroneously) decrease the Select URL budget.

4.3 Queue Timing Attack

An attacker who wants to minimize the number of network requests and calls to `selectURL` could take advantage of the queue for Shared Storage API operations from an origin. This queue is the same for both the webpage JavaScript and the worklet. Outside the worklet, calls to functions such as `sharedStorage.set` return as soon as they have been successfully added to the queue. If code inside of the worklet simultaneously makes calls to these functions, it can result in delays in when the calls from outside of the worklet are successfully added to the queue.

Writing a Value: This step remains the same as earlier attacks, simply writing to shared storage as usual.

Reading a Value: Inside the worklet, the code either does nothing or floods the queue for predetermined time intervals to represent data that the attacker wants to exfiltrate. The attacker can use existing network communication schemes for encoding (and decoding) this information.

Outside of the worklet, the attackers can, e.g., look for spikes in the number of returned function calls in a given time interval to reveal when the worklet was or was not flooding the queue. The time between those spikes reveals the behavior inside the worklet. For instance, if the worklet were trying to convey a value of zero, it could simply not do anything, and the JavaScript of the webpage would see the baseline number of function calls being returned. If, on the other hand, the worklet saw a value of one, they could flood the queue. The webpage JavaScript would see fewer of their requests returning over a given period of time, which would lead that JavaScript code to be able to infer the value in shared storage is a one.

As a concrete example, Figure 5a shows the return times for 5,000 calls to `window.sharedStorage.set` without the worklet calling the same function. Meanwhile, Figure 5b shows the return times for the same number of calls in the scenario where the code in the worklet alternates between two seconds of flooding the queue with calls to `window.sharedStorage.set` and two seconds of not taking any action. To read a value the attacker must trigger code to run in the worklet so that they can make the necessary function calls from inside the worklet. While code is running inside the worklet, the attacker starts flooding the queue with calls to `window.sharedStorage.set` from outside the worklet and recording the return times of those calls.

Practicality: Compared to attacks discussed earlier in the paper, the attack on the queue is less directly controllable. While a one-second delay in the network timing attack almost directly corresponds to a one second delay in when the server receives the response, that correlation is not quite as direct for the queue timing attack. As can be seen in Figure 5b a two second period of flooding the queue does not necessarily match with a two second period of delayed return times. Thus, an attacker would have to be careful about potentially flooding the queue again before it has had a chance to get cleared, and fail to transmit information in the process.

Another approach that an attacker may have more control over is to break up the information that they want to leak into smaller chunks. Rather than leaking all information in a single call to `selectURL`, they could make a call for each bit with a delay in between, to allow pending API calls to finish. Both of these methods would likely be much slower than other attacks but would still be able to leak at least as many bits as is allowed by the current per page budget without any budget deduction.

Potential Mitigations: Limiting the run time of functions in the worklet could limit how much information is leaked for a given call that triggers code in the worklet. However, multiple sequential calls to trigger code in the worklet could circumvent this. Limiting the total number of calls to functions that interact with the shared storage queue or rate limiting these requests could also help.

Making the return times of API calls outside of the worklet independent from what happens in the worklet would be necessary for fully preventing this avenue for leaking information. Having initial queues that are separate for the worklet and other JavaScript may help but would require additional overhead.

While Google acknowledged our report, and discussed several of the above mitigation strategies, they also commented that "while this particular side-channel could be mitigated, not all [side channels] can and we potentially need to lean on after-the-fact analysis to detect these patterns and adapt over time." While potential technical mitigations for this attack were mentioned, they have not been implemented. On the other hand, Google does appear to be

moving forward with data collection that would help with the more reactive, instead of proactive, after the fact forensics.[6]

4.4 iframe Caching Attack

While the attacks above can be carried out without decreasing the Select URL budget, they have downsides such as requiring a large number of calls to Shared Storage functions. By taking advantage of a caching covert channel on iframes, an attacker can learn what URL a call to `selectURL` returned in the past for that top-level page without decreasing the Select URL budget or making any calls to Shared Storage API functions after the writing phase. This attack mainly takes advantage of the fact that in the normal use of the `selectURL` function to load a resource into an iframe, that resource can be cached, and it is possible to determine if those resources have been cached without interacting with the Shared Storage API.

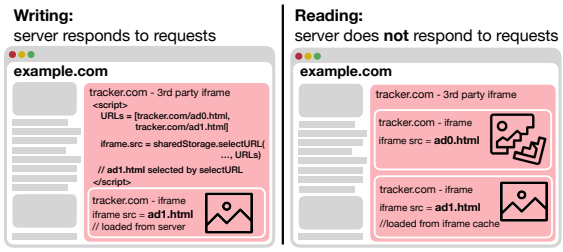


Figure 6: Writing a value from shared storage to the iframe cache and reading it on a subsequent visit to the page.

Writing a Value: For this attack, we assume that an identifier has already been written to shared storage in the same way as in earlier attacks and that the attacker now wants to store this identifier in a way that does not require the Shared Storage API to retrieve it. To accomplish this, the page makes a call to `selectURL`, passing multiple URLs as described in Figure 2, and loads the selected resource into an iframe. For the sake of simplicity, let us assume that two URLs were passed in `ad0.html` and `ad1.html` and `ad1.html` was selected and loaded into an iframe. `ad0.html` would not be loaded into an iframe and would not be cached. The server should respond as usual, resulting in the selected URL being loaded *and cached* for that top-level page (if the URL were to be subsequently requested from a different top-level page it would not be in the cache for that top-level page).

Reading a Value: In the reading phase, the page attempts to load both `ad0.html` and `ad1.html` by setting the `src` attribute of an iframe to each of those URLs. This does not involve any calls to any Shared Storage API functions and therefore no deduction from the Select URL budget.

The server should *not* respond to requests in this phase so that only resources that have previously been cached would be loaded without an error. Figure 6 shows the result of this process: only `ad1.html`, which had previously been cached, is loaded.

[6]In fact, part of their data collection efforts seems to be driven by our disclosure of the fact that calling `selectURL` with a single URL can circumvent the privacy budget in https://github.com/WICG/shared-storage/issues/86

Since these nested iframes are from the same domain (`tracker.com`) communication with the top-level `tracker.com` iframe is possible, allowing the attacker to determine what resource was loaded from the cache. This essentially provides the same information as previous calls to `selectURL` without the Select URL budget deduction since in both cases `ad1.html` was loaded into the page and `ad0.html` was not.

This reading process is similar to reading an identifier based on favicon caching as described in prior research [69].

As Chrome partitions the iframe cache per top-level site, this circumvention method is only helpful to a tracker if a user revisits a site and the same iframe is loaded onto the page. Previous studies have shown revisitation rates of top-level pages to be 60% to 78% within a 20-day period [15, 86]. Paired with the fact that popular trackers appear on a large percentage of websites [26] there is a good chance that those trackers could take advantage of iframe caching. Attackers would also need to be sure that they do not accidentally poison their cache by loading the resource outside of a call to `selectURL`. In our Figure 6 example, if the server ever loaded `ad0.html` into that iframe, it would be cached, and *both* `ad0.html` and `ad1.html` would be loaded from the cache in the reading phase and the attacker would not know which one was added to the cache by the call to `selectURL`.

To summarize, this caching of resources loaded into iframes allows an attacker to replay calls to `selectURL` across reloads of the page without a deduction from the Select URL budget.

Practicality: The speed bottleneck for this attack is writing the identifier. As this attack requires calls to `selectURL` with more than one URL writing the identifier is constrained by the Select URL budget and the fact that the iframe cache is partitioned by top-level page. How long writing an identifier to a cache would take depends on how many sites the tracker wants to move their identifier from shared storage into the cache and how often a user revisits these top-level sites. If the tracker were to focus on writing a 33 bit identifier to the iframe cache from shared storage for a single top level site and the user was to visit the page every day after the tracker's Select URL budget resets, this would have to take place over three days given the current 12 bit per day budget. That being said, on the second and third day the bits that had already been cached could be read and used in conjunction with other methods (e.g., browser fingerprinting) to identify the user.

The reading process of this attack only involves making normal network requests to a server without having to invoke aspects of the Shared Storage API that would involve budget deductions. This means that reading a long value in this way does not rely on a user repeatedly visiting a page. If we use the setup described above, where loading one resource represents a value of zero and the other represents one, this would require 66 requests to read a 33 bit identifier as the two possible resources must be unique for each bit of the identifier. To the extent that the browser is capable, these requests can happen in parallel. The median number of requests made by a webpage at the beginning of 2025 was over 70 [38], so an additional 66 would be a proportionally large increase but, in the reading phase the server does not have to return any content.

In terms of storage, this attack requires caching a resource for each bit of the identifier. Still, the resources themselves could consist

of merely a text file with a single character in it making the impact on the amount of content stored in the cache relatively small.

Potential Mitigations: Preventing resources loaded from the result of a call to `selectURL` from being cached or loaded into iframes could be avenues to mitigating this particular issue. By taking these steps, the API would prevent "free" API calls when a user revisits a page that had previously made a call to `selectURL`. Google plans to address this class of attacks by preventing the results of `selectURL` from being loaded into iframes and requiring that the URLs be loaded into a fenced frame. Currently, this change is not planned to take effect until 2026 at the earliest. Moreover this, ties the mitigation to the effectiveness and the adoption of fenced frames.

4.5 Data Leakage Over Time Attack

Since the Select URL budget periodically resets (every 24 hours), additional data can be leaked over time, in stages. This means the Shared Storage API is susceptible to a vulnerability similar to one observed in the FLoC proposal [6, 75], which we summarize below. This issue of arbitrary amounts of data being leaked over time has long been acknowledged by the developers of the API as an inherent issue with any proposal where the limits on how much data can be leaked reset after a given amount of time [55].

In FloC users were sorted into "cohorts" with similar browsing histories. After every time period (initially 7 days), users would be sorted into a new cohort. Websites could read what cohort a user belonged to, but since there were many users in each cohort, the idea was that the users could not be uniquely identified. Unfortunately, research showed that over time the sequences of cohorts that a user was assigned to were often unique [6]. To derive that sequence of cohorts, a (third-party) attacker could take advantage of various types of partitioned storage that are accessible, even if third-party cookies are not. A similar attack can be used against the Shared Storage API. Specifically, an attacker can read different parts of their chosen identifier every time the budget resets.

Writing a Value: Writing an identifier is identical to the network timing attack or crashing attack. The attacker simply writes values to shared storage either with JavaScript or through network response headers.

Reading a Value: When the Select URL budget resets for a given page the attacker reads the next few bits of the identifier that they have previously stored in shared storage, by calling `selectURL`, and storing the bits that were leaked based on what URL was requested in any form of partitioned storage they have access to. This moves the identifier that was stored in shared storage, and is therefore the same across all top-level pages, into storage that is partitioned by the top-level page but has no limit on the frequency at which it can be read. Over time this means that the partitioned storage for a tracker on each top-level page will have the same identifier stored for a user. This is similar in concept to the iframe caching attack in which the calls to `selectURL` are essentially reused across reloads of a single top-level site.

Differences from FLoC Attack: With the Shared Storage API an attacker can have more control over making an identifier unique to a given user, rather than relying on the cohorts FLoC assigns the user to based on their browsing history. With each call to `selectURL` the attacker can choose to read a new part of the identifier, while new information can only be gained from FLoC when the user is assigned to a new cohort.

The Shared Storage API has the privacy benefit that for a given time period there is a limit on how often even part of an identifier can be read, whereas for FLoC, reading which cohort a user has been assigned to does not suffer from the same limitation.

Practicality: This attack does not involve deviating from the intended use of the Shared Storage API or the `selectURL` function so, in this case, the Select URL budget is the limiting factor in how much data can be leaked. This means that only 12 bits of data can be leaked in this way per day. Individual calls to `selectURL` are negligibly fast when the code that is run inside the worklet is simple. A single call to `selectURL` is intended to leak up to three bits of information due to the limit on the number of URLs that can be passed in to the function. Three calls to `selectURL`, which would leak 12 bits, could easily take place in far less time than even the lowest estimates for median dwell time on a page [36, 47, 52, 53]. As discussed in relation to the iframe caching attack, users frequently revisit sites so it is likely that attackers could continue to leak information as users revisit these sites over time.

Potential Mitigations: The Select URL budget is an integral part of the Shared Storage proposal because some information is inherently leaked when `selectURL` is called. Lowering the budget would slow down the rate at which the data could be leaked but would not stop it. Limiting even partitioned storage methods could help to prevent third parties from building up the identifiers but would impact far more than just the Shared Storage API. If fenced frames were required and did not leak the result of `selectURL` to the server the embedding frame could not build up an identifier, but this requires many changes to the status quo and would make using the API for anything other than static content very difficult.

4.6 Google's Responses

Attack	Status	Link	Location of Disclosure
Network Timing	Proposed Solution	https://github.com/WICG/shared-storage/issues/86	Initial Issue
Crash	Partially Fixed	https://github.com/WICG/shared-storage/issues/86	Issue Thread
Queue Timing	Proposed Solution	https://github.com/WICG/shared-storage/issues/136	Initial Issue
iframe Caching	Planned Fix	https://github.com/WICG/shared-storage/issues/86	Initial Issue
Data Leakage Over Time	No Planned Fix	https://github.com/WebKit/standards-positions/issues/10	Issue Thread

Figure 7: Details of Google's responses to disclosures of possible attacks.

For each attack described in this paper Google directly replied discussing potential fixes or acknowledging the inherent limitation of the API, as was the issue of the amount of data that can be leaked over time. For two of our disclosures, Google's response also referenced relying on identifying indications of misuse (e.g., making a large number of requests or intentional crashes) to identify misuse of the API. Recently, Google added more logging capability to the implementation of the API, which references our GitHub issue for the Network Timing Attack, to help identify cases where many calls to `selectURL` with a single URL are made. Disclosures as well

as Google's responses to all of the attacks can be accessed at the links in Table 7.

5 Usage of the Shared Storage API

To understand the adoption of the Shared Storage API, we perform multiple measurements across different aspects of the API: the proportion of domains that have completed the attestation process, both among popular and tracking domains, and the prevalence of Shared Storage API usage in the wild. These measurements provide insight into users' current exposure to the API as well as provide an overview of the characteristics of companies that have access to the API and would therefore be able to exploit the API if desired.

5.1 Attestation Completion

As discussed in Section 2, companies must complete Google's attestation process before being able to use the Shared Storage API. Understanding which companies have completed this process directly relates to who could carry out these attacks as well as attacks on several other APIs in the Privacy Sandbox ecosystem.

To compile a list of sites that may have completed the attestation process we used two sources: Google's Privacy Sandbox Enrollment Report, a public list of what companies have completed the enrollment process which was last updated on June 28th, 2024,[7] and a list of domains shipped with Google Chrome titled `privacy-sandbox-attestations.dat` from December 25th, 2024 from which we extracted all plaintext domains.

After excluding URLs used exclusively for demos of the Privacy Sandbox API, provided by Google, we were left with 289 domains. Google requires that the attestation files be hosted at a `well-known` URL and available to researchers. In late December 2024, for each domain, we used the Python `request` library to make a request to the path on the domain where the attestation file should be hosted. For domains where this process did not successfully retrieve a valid attestation file we manually visited the URL in an attempt to retrieve the file. From this process we were able to obtain 245 valid attestation files, meaning that 85% of the domains that we checked were hosting an attestation file.

During the process of getting access to these APIs from Google, developers can request access to all or a subset of the APIs that require an attestation. Of the attestation files we collected, 61% (149) reported that the domain had access to the Shared Storage API. With public information, it is only possible to know if the domain was ultimately granted access to the Shared Storage API; on the other hand, no information is available about requests Google may have denied.

5.1.1 Popularity of Domains With Shared Storage Access. To understand the popularity of the domains, which we use as a proxy for companies, that have completed the attestation process we check the rank of these domains in the Tranco list[8] [50] from the same date as the Chrome browser file. Figure 8 shows the cumulative proportion of domains from this list that we found hosting an attestation file and had access to the Shared Storage API specifically. 6.4% of the top 1,000 domains host valid attestation files while 4.3% host valid attestation files that include the Shared Storage API.

[7] https://github.com/privacysandbox/attestation/blob/main/enrollment_report.csv
[8] The Tranco list used here can be found at https://tranco-list.eu/list/932Q2

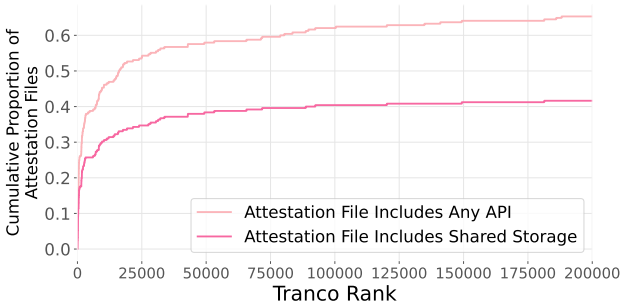


Figure 8: The cumulative proportion of attestation files with a domain in the Tranco top 200K.

In short, while a relatively small number of domains have completed the attestation process, at least some companies are aware of the Privacy Sandbox APIs, including the Shared Storage API, and have made efforts to secure access to them.

5.1.2 Tracking Affiliation of Domains With Shared Storage Access. While comparing the domains that have completed the attestation process to the Tranco list provides insight into the popularity of the domains with access to the API, Shared Storage should be primarily useful to advertisers. Companies associated with advertising and tracking may have more of an incentive to exploit Shared Storage for tracking users.

To get a sense of how many of the companies with attestation files are associated with these areas we checked the domains with attestation files, which we previously compared to the Tranco list, against tracking-related domains from the Disconnect Tracker Protection list [19]. Of the domains that have a valid attestation file 48.2% appeared in the Disconnect list. Similarly, 56.4% of the domains with access to the Shared Storage API appeared in the Disconnect list.

From the other direction, out of all of the companies listed in the disconnect file, 112 (7.2%) have at least one URL that completed the attestation process and 78 (5.0%) of companies had at least one URL that had completed the process and been granted access to Shared Storage. When we only consider companies that the list designates as related to advertising, we see that 9.2% (104) had completed the attestation process and 6.5%(73) of those companies had been granted access to Shared Storage, both of which are slightly higher percentages than in the list as a whole. Altogether, this shows us that many of the companies that have gone through the attestation process are associated with tracking, but there are still many known tracking domains that do not have access to the API.

5.2 Shared Storage in the Wild

To better understand users' exposure to the Shared Storage API we conducted an additional crawl in early February 2025 to look for usage of the API on popular websites. This is necessary because, even if a website has not completed the attestation process themselves, it may include content from a domain that has. Companies that complete the attestation process may also choose not to use the APIs that they have access to.

For each of the top 10,000 domains and a random sample of 5,000 domains from the remaining top 1 million domains from the Tranco list of popular domains[9] we visited the page in a fresh instance of Puppeteer with the Shared Storage API enabled. Through the Chrome developer tools, we captured all instances of the API being called both outside and within the worklet.

From this crawl, we found that approximately 10.1% (1,311) of the 12,955 sites we were able to reach did invoke the Shared Storage API in some way. Most relevant to our attacks, 1,250 of these pages (9.6%) made at least one call to `selectURL`, with 1,613 calls to `selectURL` across all pages.

All of these interactions can be traced back to four unique domains. Two domains associated with Google, `securepubads.g.doubleclick.net` and `ep3.adtrafficquality.google`, were the most prolific in their use of the API, using Shared Storage on 1,222 and 177 websites respectively. `crcldu.com` (46 websites) and `ads.optable.co` (17 websites) made up the other domains responsible for these instances.

The reality that the Shared Storage API is being used on popular websites indicates that end users are already being exposed to the API. In the course of these measurements, we did not look for any evidence of the attacks described in the paper being currently exploited on websites. However, historically, fingerprinting companies have used a variety of methods to learn about users [48]. Additionally, attacks that were initially discussed theoretically, such as canvas fingerprinting [58], have later been found to be used in practice [1, 21, 48]. While the adoption of code that calls functions that read from Shared Storage remains fairly low and originates from a small set of domains, it does occur, and many more companies have completed the process necessary to start using the Shared Storage API if they choose to in the future.

6 Discussion

In this section, we discuss the limitations of our study and possible future work (Section 6.1), ethical considerations (Section 6.2), implications for user choice (Section 6.3), the positions of other browsers (Section 6.4), and the impact on cross browser compatibility and centralization (Section 6.5).

6.1 Limitations and Future Work

Shared storage is a complicated proposal that depends on several other technical proposals (e.g., fenced frames and the Private Aggregation API), and its feature set and complexity is continuing to grow over time. With the continued evolution of the API and the other proposals that it depends on new issues might be introduced. In this paper, we identify several exploitable issues in the Shared Storage API as designed and implemented today but there may be more attacks that are possible than we identified. If Google were able to fix all of the issues we laid out in the paper, the API would undergo significant changes, including to functionality, and would require additional analysis. More generally, our paper highlights the need for a more principled API design, since these APIs introduce new covert channels, rather than post-hoc fixes.

Unfortunately, our findings are not isolated events. With the similarities and interactions between different privacy sandbox proposals, we expected (1) new proposals could be susceptible to the same attacks that have been identified in previous privacy sandbox proposals (e.g., Protected Audience API is susceptible to similar timing attacks as the worklet for the Shared Storage API [73]) and (2) interactions between the APIs may introduce vulnerabilities to the privacy protections the individual proposals are attempting to provide (e.g., the new queue introduced by the Shared Storage API itself introduces a new covert channel that can be abused to violate privacy). Future work could look for additional attacks that updates to the API introduce or for new opportunities to improve both Shared Storage and similar proposals in the Privacy Sandbox.

Our measurement of the usage of the Shared Storage API in Section 5 has limitations. If is possible that if a real user were to interact with the same sites that were a part of our measurement they could cause additional code to run that could trigger calls to the Shared Storage API or avoid some level of bot detection. In practice, this means that the true rate at which the API is used on websites could be higher than what was found in our measurement. Further, the measurements presented in this paper are only snapshots in time, with the changes in Google's position on third-party cookies this use may vary.

While other work has addressed the prevalence of other privacy sandbox APIs on the Internet [10], as new or less studied APIs crop up, further analysis may be warranted to understand the scope of how the APIs are being used and interact. While our study did not look for potential misuse of the APIs, looking for these types of usage across Privacy Sandbox APIs could be very beneficial to users, considering Google's comments on addressing privacy violations due to Shared Storage as they appear.

Covert channels frequently have their limitations and the attacks proposed in this paper are no exception. Variability in network conditions could cause issues for the network timing attack, such as abnormally high RTTs being misinterpreted as a delay. More generally, interference from factors outside of an attacker's control on any of these covert channels may lead to an attacker not being able to accomplish their goal. Many attacks also rely on the user actively remaining on a page for a period of time. If users spend only a few seconds on a page, it is unlikely that an attacker would be able to exfiltrate the entirety of an identifier.

If attacks start to noticeably degrade a user's browsing experience, such as the browser slowing down due to extensive use of the API, or delays in legitimate network requests due to a large number of requests being sent in an attempt to exfiltrate data, the user may take steps such as reloading a page or navigating away from a tab. This could interrupt some of the attacks presented in this paper.

6.2 Ethics

As the Shared Storage API is currently implemented and available for use through the Chrome browser, it was necessary that we report these issues. Therefore, we disclosed the attacks presented in this paper to the Shared Storage API team through several issue reports to the W3C GitHub repository for the project.[10]

[9] The Tranco list used for this crawl is the same one as used in Section 5.1.1.

[10] https://github.com/WICG/shared-storage/issues/86
https://github.com/WICG/shared-storage/issues/136

All experiments were conducted from our own machines, on ourselves, and on self-hosted websites under our control. We deliberately used local exceptions to the attestation requirement rather than going through the formal attestation approval process, as the attestation process requires agreeing not to use the API for the identification of users across websites. This also avoided the issue of requiring Google to review an attestation form submitted by us, which could have unnecessarily slowed down the review for any other companies that were simultaneously completing the attestation process. Our approach also had the advantage of ensuring that the scripts for identifying users on our websites would not impact anyone who accidentally visited the site, as they would presumably not have set up the custom exceptions.

6.3 User Choice

Currently, there does not seem to be a way to individually disable the Shared Storage API in the normal settings menus of Chrome as of version 128. At present, disabling third-party cookies, disabling all privacy sandbox APIs, or experimental flags appear to be the only available methods for users to prevent sites from using the Shared Storage API. There has been some discussion of adding a dedicated setting for the API, but it has been wrapped into the general Privacy Sandbox Setting which does not explicitly mention the Shared Storage API.[11] Providing similar levels of control for the API as currently exist for third-party cookies would allow users to make choices about their usage of the API if they are uncomfortable with individual proposals within the set of privacy sandbox proposals. As with any technical setting, explaining what it is to a general audience in an accurate and accessible way would require significant effort as previous work has shown [37, 60, 84]. As a result of the UK Information Commissioner's Office raising similar concerns about the lack of end user facing information about how the API can be used, Google is planning on making unspecified changes to their user interface [13].

More generally, with the move towards alternatives to cookies, existing tracking protection tools that empower users to make choices about data storage and currently only focus on cookies will need to adapt to these new APIs to continue to provide users with the same level of protection in the absence of existing methods for users to manage their data. Current consent management platforms will also need to adapt to these new proposals as legal consent requirements (e.g., GDPR) can also apply to data stored and accessed via these APIs [27].

Since the Shared Storage API has garnered less public scrutiny than other other privacy sandbox proposals (e.g., FLoC and Topics) it is unlikely that even fairly tech-savvy users are aware of this new API and the potential implications for their privacy.

6.4 Industry Positions

Other browser vendors have also expressed their concerns about the Shared Storage API and its ability to preserve user privacy. Apple [55] expressed concerns that "sites that users visit often would be able to extract an arbitrary amount of data over time" due to the resetting budget and the dependence on other proposals such as fenced frames which, at the time, did not have full specifications. Mozilla's negative position on the proposal was that they had "significant concerns about the viability of the isolation components of this design," although they did not cite any specific attacks against the API. They also stated that the use cases for the API did not "justify the complexity and privacy risks" [23, 56]. While these other major browsers have not implemented the API, Shared Storage is already deployed in Chrome and therefore in play for many users.

[11] https://github.com/WICG/shared-storage/issues/9

6.5 Compatibility and Centralization

As mentioned in Section 2, Google had added the requirement that companies complete an attestation process before being allowed to use privacy sandbox APIs including the Shared Storage API.

While companies may follow this agreement, there is always the chance that they will surreptitiously renege on their word and use the API for tracking. Having companies publicly attest that they will not use the API to re-identify users could potentially open the companies up to action from the Federal Trade Commission in the United States for deceptive practices or at least public backlash if they are found to be breaking that agreement.

While having platform specific review processes has precedent in the app ecosystem (e.g., Apple App Store Review for iOS apps) this goes significantly further forcing any website developer that wants to use the Privacy Sandbox APIs to go through this process. This provides extra friction for companies trying to take advantage of the privacy benefits that these APIs provide over technologies like third-party cookies. As Google is presumably reviewing these submissions, it makes them a gatekeeper for who gets to use the API. Since there are no practical techniques for detecting privacy leaks via covert channels, this also positions Google as the sole, arbitrary executioner. With Google also being a large player in the advertising space, what this means for competition is unclear.

Going against the opinions of other major browser vendors, Google has moved forward with the Shared Storage API. Were this API to majorly take off this would force website developers to customize their websites even more based upon which browser a user is visiting their website from.

7 Related Work

We introduce related work by first discussing relevant steps that are being taken by various stakeholders to address tracking. We then explain how these mitigation strategies can break down and become avenues for tracking in themselves sometimes in similar ways to the Shared Storage API. Finally, we discuss the deployment of Google Privacy Sandbox APIs.

7.1 Tracking and Advertising Mitigations

Many steps have been taken to prevent the potential negative impacts on users of tracking and advertising. Browser extensions like Privacy Badger and uBlock Origin have been developed to prevent tracking and/or stop ads from being displayed through blocking/modifying network requests and cookies. These tools have become widely adopted. A 2023 report from eye/o found that 912 million users were actively using ad-blockers [24].

Many browsers have even started to roll out their own forms of built-in protections. Their strategies take various forms such as limiting the use of third-party cookies, disabling APIs that are

primarily used for tracking, and more [14]. In Firefox, many of these efforts are bundled under the umbrella of the "Enhanced Tracking Protection" setting, Safari has "Intelligent Tracking Protection," and Brave has "Shields". As discussed, Google has been moving forward with its "Privacy Sandbox" proposals, which in addition to the Shared Storage API, includes proposals such as federated learning of cohorts (FLoC) [30], the Topics API [34], Protected Audience API [33], and Aggregate Reporting API [32]. Each of these browsers has a different approach and they vary in how strict the protections they offer are, with each having different tradeoffs. Unlike browser extensions that require users to go out of their way to install them, tools that are built into browsers are frequently rolled out to users without their intervention.

There is also extensive development and research on additional methods of preventing tracking and making the advertising ecosystem more privacy preserving, while still enabling the benefits that users find in being shown relevant ads. Some of these proposals are focused on measuring ad conversions and effectiveness [18, 42, 71, 85, 87], allowing ad networks to charge advertisers without knowing what ad was shown [66, 74], and more [81].

7.2 Privacy-Enhancing Technologies as Tracking Vectors

Unfortunately, sometimes the very steps taken to prevent tracking can then become vectors used by companies to re-identify users. This paper focuses on how that is possible for one specific tool, but the Shared Storage API is far from the only privacy-enhancing tool that can be misused in this way. For example, the presence of the Do Not Track header, which was originally an attempt to allow users to opt out of tracking, can also be used for browser fingerprinting [20, 61]. Having privacy preserving (or other) extensions installed may also make a user more identifiable [49]. Similarly, accessing a website via a less widely used but more privacy-preserving browser or having extraneous APIs and features disabled can also factor into a browser fingerprinting profile for a user [20]. User-Agent Client Hints were intended to allow access to User-Agent string information in a more privacy-preserving way but were still found to be used for tracking [65]. Shared Storage could become a similar telltale since a Chrome user disabling the API in their browser would look different than a user with the API enabled.

Many of the Google Privacy Sandbox proposals have been shown to have flaws that leak more information than intended, which could lead to the re-identification of users. One of the earliest Privacy Sandbox proposals to draw public scrutiny was the FLoC proposal. For each block of time, FLoC assigned users into *cohorts* based on the similarities of their browsing histories. Several papers have demonstrated how the sequences of *cohorts* users would be assigned to could be used to re-identify those users over time [6, 75]. In Section 4.5 we showed that data stored in Shared Storage can similarly be used to build up identifiers across websites.

The successor of FLoC, the *Topics API*, had the users' browsers learn the interests of the users and expose those interests through the API rather than have advertisers track cohorts across the Web to learn the interests of the groups. The Topics API was similarly shown to not prevent the re-identification of users [3, 7, 40, 41]. Despite this, the API continues to be a part of Chrome much like the Shared Storage API. Academics are not the only ones to have raised concerns about these proposals, which have also drawn scrutiny from industry [68, 72, 73] as well as privacy advocate groups like the EFF [16, 46] and the Competition and Markets Authority in the UK [12]. Further works have also sought to accurately assess the privacy risks of the API to users [8, 11, 22].

The Protected Audience API facilitates on-device ad auctions by the browser. Part of this process involves the execution of code in an isolated environment that is susceptible to timing attacks [73]. As we described in Section 4.1, the Shared Storage API has a similar concept of running code in an isolated environment, and is also susceptible to a similar type of timing attack. Many other limitations of the privacy guarantees of the Protected Audience API have also been shown [2, 10, 54, 73]. Our work falls within this line of research, showing how the privacy-preserving aspects of the Shared Storage API can still be circumvented to the detriment of users.

7.3 Privacy Sandbox Deployment

Privacy Sandbox APIs are actively being used by companies on websites that users frequent [10, 43, 44, 62]. Johnson et al. found that in 2023 the Topics API was used on over 35% of the approximately 60K websites they surveyed [44]. The use of these APIs is also not uniform across all of the Privacy Sandbox, with less popular APIs such as the Protected Audience API being used on less than 10% of the surveyed sites [44]. Chrome Status metrics for the Privacy Sandbox APIs also show a similar trend, with the Protected Audience API being used on less than 6% of Chrome page loads at the beginning of 2025 and the method that retrieves information from the Topics API being used on over 11% of Chrome page loads in the same time period [29]. In Section 5.2 we similarly described the adoption of the Shared Storage API on popular websites which, to the best of our knowledge, has not been previously reported. Prior work has also shown that well known advertisers are active users of the Privacy Sandbox APIs [44, 62] and often early adopters [78], which we find to also be true for the Shared Storage API (Section 5).

8 Conclusion

In this paper, we have shown the Shared Storage API is vulnerable to several covert channels that undermine the stated privacy goals of the API. These covert channels have the potential to allow users to be re-identified by third parties across websites. We have notified Google of the vulnerabilities in the API, but several attacks currently remain feasible in Chrome, and are not likely to be addressable without rethinking the API from first principles. Creating APIs that are more privacy preserving than existing technologies can greatly benefit users but can be very difficult to get right. And, while the Shared Storage API does take a step towards this goal, it also introduces new complexities and unknowns that detract from the potential benefits it offers.

Acknowledgments

This material is based upon work supported by the National Science Foundation Grant No. 2120642 and the National Science Foundation Graduate Research Fellowship under Grant No. DGE2140739. Additionally, we thank Eric Zeng, Arjun Brar, Ariana Mims, and Hanan Hibshi for their advice and feedback on this project.

References

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM Conference on computer and communications security*. 674–689.

[2] Mir Masood Ali, Binoy Chitale, Mohammad Ghasemisharif, Chris Kanich, Nick Nikiforakis, and Jason Polakis. 2023. Navigating Murky Waters: Automated Browser Feature Testing for Uncovering Tracking Vectors. In *Network and Distributed System Security (NDSS) Symposium*.

[3] Mário S. Alvim, Natasha Fernandes, Annabelle McIver, and Gabriel H. Nunes. 2024. The Privacy-Utility Trade-off in the Topics API. In *Proceedings of the 2024 on ACM Conference on Computer and Communications Security (CCS '24)*. Association for Computing Machinery, Salt Lake City, UT, USA, 1106–1120. doi:10.1145/3658644.3670368

[4] Anthony Chavez. 2024. A new path for Privacy Sandbox on the web Share . https://privacysandbox.com/news/privacy-sandbox-update/.

[5] Joshua Asplund, Motahhare Eslami, Hari Sundaram, Christian Sandvig, and Karrie Karahalios. 2020. Auditing Race and Gender Discrimination in Online Housing Markets. *Proceedings of the International AAAI Conference on Web and Social Media* 14, 1 (May 2020), 24–35. doi:10.1609/icwsm.v14i1.7276

[6] Alex Berke and Dan Calacci. 2022. Privacy Limitations of Interest-based Advertising on The Web: A Post-mortem Empirical Analysis of Google's FLoC. In *Proceedings of the 2022 ACM Conference on Computer and Communications Security (CCS '22)*. ACM, Los Angeles, CA, USA, 337–349. doi:10.1145/3548606.3560626

[7] Yohan Beugin and Patrick McDaniel. 2024. Interest-disclosing Mechanisms for Advertising are Privacy-Exposing (not Preserving). *Proceedings on Privacy Enhancing Technologies* (2024).

[8] Yohan Beugin and Patrick McDaniel. 2024. A Public and Reproducible Assessment of the Topics API on Real Data. In *2024 IEEE Security and Privacy Workshops*. 1–8. doi:10.1109/SPW63631.2024.00005

[9] Brave Help Center. 2024. How Do I Manage Cookies In Brave? https://support.brave.com/hc/en-us/articles/360050634931-How-Do-I-Manage-Cookies-In-Brave.

[10] Giuseppe Calderonio, Mir Masood Ali, and Jason Polakis. 2024. Fledging Will Continue Until Privacy Improves: Empirical Analysis of Google's Privacy-Preserving Targeted Advertising. In *33rd USENIX Security Symposium*. USENIX, Philadelphia, PA, 4121–4138. https://www.usenix.org/conference/usenixsecurity24/presentation/calderonio

[11] CJ Carey, Travis Dick, Alessandro Epasto, Adel Javanmard, Josh Karlin, Shankar Kumar, Andres Muñoz Medina, Vahab Mirrokni, Gabriel Henrique Nunes, Sergei Vassilvitskii, et al. 2023. Measuring re-identification risk. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

[12] Competition and Markets Authority. 2024. Investigation into Google's 'Privacy Sandbox' browser changes. https://www.gov.uk/cma-cases/investigation-into-googles-privacy-sandbox-browser-changes.

[13] Competitions & Markets Authority. 2024. CMA Q1 2024 update report on implementation of the Privacy Sandbox commitment. https://assets.publishing.service.gov.uk/media/662baa3efee48e2ee6b81eb1/1._CMA_Q1_2024_update_report_on_Google_Privacy_Sandbox_commitments.pdf.

[14] Cookie Status. 2024. Cookie Status. https://www.cookiestatus.com.

[15] Kyle Crichton, Nicolas Christin, and Lorrie Faith Cranor. 2021. How Do Home Computer Users Browse the Web? *ACM Trans. Web* 16, 1, Article 3 (Sept. 2021), 27 pages. doi:10.1145/3473343

[16] Bennett Cyphers. 2021. Google's FLoC Is a Terrible Idea. https://www.eff.org/deeplinks/2021/03/googles-floc-terrible-idea.

[17] Amit Datta, Michael Carl Tschantz, and Anupam Datta. 2015. Automated Experiments on Ad Privacy Settings: A tale of opacity, choice, and discrimination. *Proceedings on Privacy Enhancing Technologies* 2015, 1 (2015), 92–112.

[18] John Delaney, Badih Ghazi, Charlie Harrison, Christina Ilvento, Ravi Kumar, Pasin Manurangsi, Martin Pál, Karthik Prabhakar, and Mariana Raykova. 2024. Differentially Private Ad Conversion Measurement. *Proc. Priv. Enhancing Technol.* (2024).

[19] Disconnect. 2024. Tracker Protection lists. https://github.com/disconnectme/disconnect-tracking-protection/blob/master/services.json.

[20] Electronic Frontier Foundation. [n. d.]. Cover Your Tracks. https://coveryourtracks.eff.org.

[21] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM Conference on Computer and Communications Security (CCS '16)*. ACM, Vienna, Austria, 1388–1401. doi:10.1145/2976749.2978313

[22] Alessandro Epasto, Andres Munoz Medina, Christina Ilvento, and Josh Karlin. 2022. Measures of cross-site re-identification risk: An analysis of the Topics API Proposal. https://raw.githubusercontent.com/patcg-individual-drafts/topics/main/topics_analysis.pdf.

[23] Eric Trouton. 2022. Request for Mozilla Position on Shared Storage. https://github.com/mozilla/standards-positions/issues/646.

[24] eye/o. 2024. 2023 eyeo Ad-Filtering Report. https://info.eyeo.com/adfiltering-report.

[25] Justas Gaubys. 2023. Most popular web browsers in 2023. https://www.oberlo.com/statistics/browser-market-share.

[26] Ghostery. 2021. Tracking the Trackers 2020: Web Tracking's Opaque Buisness Model of Selling Users. https://www.ghostery.com/blog/tracking-the-trackers-2020.

[27] Google. 2023. Privacy-related compliance FAQs. https://developers.google.com/privacy-sandbox/overview/privacy-compliance-faqs.

[28] Google. 2024. Update on the plan for phase-out of third-party cookies on Chrome Share. https://privacysandbox.com/intl/en_us/news/update-on-the-plan-for-phase-out-of-third-party-cookies-on-chrome/.

[29] Google. 2025. Privacy Sandbox metrics. https://pscs.glitch.me.

[30] Google Privacy Sandbox. 2021. FLoC. https://developers.google.com/privacy-sandbox/archive/floc.

[31] Google Privacy Sandbox. 2022. Fenced frames overview. https://developers.google.com/privacy-sandbox/relevance/fenced-frame.

[32] Google Privacy Sandbox. 2022. Private Aggregation API overview. https://developers.google.com/privacy-sandbox/relevance/private-aggregation.

[33] Google Privacy Sandbox. 2022. Protected Audience API overview. https://developers.google.com/privacy-sandbox/relevance/protected-audience.

[34] Google Privacy Sandbox. 2022. Topics API for Web overview. https://developers.google.com/privacy-sandbox/relevance/topics.

[35] Google Privacy Sandbox. 2023. Developer enrollment for the Privacy Sandbox. https://developers.google.com/privacy-sandbox/blog/announce-enrollment-privacy-sandbox.

[36] Kirstie Hawkey and Kori Inkpen. 2005. Web browsing today: the impact of changing contexts on user activity. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems (CHI EA '05)*. Association for Computing Machinery, Portland, OR, USA, 1443–1446. doi:10.1145/1056808.1056937

[37] Franziska Herbert, Steffen Becker, Leonie Schaewitz, Jonas Hielscher, Marvin Kowalewski, Angela Sasse, Yasemin Acar, and Markus Dürmuth. 2023. A World Full of Privacy and Security (Mis)conceptions? Findings of a Representative Survey in 12 Countries. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM, Hamburg, Germany, Article 582, 23 pages. doi:10.1145/3544548.3581410

[38] http archive. [n. d.]. Report: State of the Web. https://httparchive.org/reports/state-of-the-web.

[39] Basileal Imana, Aleksandra Korolova, and John Heidemann. 2021. Auditing for Discrimination in Algorithms Delivering Job Ads. In *the Web Conference (WWW '21)*. ACM, Ljubljana, Slovenia, 3767–3778. doi:10.1145/3442381.3450077

[40] Nikhil Jha, Martino Trevisan, Emilio Leonardi, and Marco Mellia. 2023. On the Robustness of Topics API to a Re-Identification Attack. *Proceedings on Privacy Enhancing Technologies* 4 (2023), 66–78.

[41] Nikhil Jha, Martino Trevisan, Emilio Leonardi, and Marco Mellia. 2024. Re-Identification Attacks against the Topics API. *ACM Trans. Web* (June 2024). doi:10.1145/3675400

[42] John Wilander. 2021. Introducing Private Click Measurement, PCM. https://webkit.org/blog/11529/introducing-private-click-measurement-pcm/.

[43] Garrett Johnson. 2024. Unearthing Privacy-Enhancing Ad Technologies (PEAT): The Adoption of Google's Privacy Sandbox. *Available at SSRN* (Oct. 2024). http://dx.doi.org/10.2139/ssrn.4983927.

[44] Garrett A Johnson and Nico Neumann. 2024. The advent of privacy-centric digital advertising: Tracing privacy-enhancing technology adoption. https://pep.gmu.edu/wp-content/uploads/sites/28/2024/04/Johnson-Neumann.pdf.

[45] Josh Karlin. 2023. Intent to Ship: Shared Storage API. https://groups.google.com/a/chromium.org/g/blink-dev/c/dZ0NRwh7cvs.

[46] Thorin Klosowski. 2023. How To Turn Off Google's "Privacy Sandbox" Ad Tracking-and Why You Should. https://www.eff.org/deeplinks/2023/09/how-turn-googles-privacy-sandbox-ad-tracking-and-why-you-should.

[47] Ravi Kumar and Andrew Tomkins. 2010. A characterization of online browsing behavior. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. Association for Computing Machinery, Raleigh, North Carolina, USA, 561–570. doi:10.1145/1772690.1772748

[48] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*. 878–894. doi:10.1109/SP.2016.57

[49] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. 2021. Fingerprinting in Style: Detecting Browser Extensions via Injected Style Sheets. In *30th USENIX Security Symposium*. USENIX, 2507–2524.

[50] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Symposium on Network and Distributed System Security (NDSS '19)*. ISOC, San Diego, California, USA. doi:10.14722/ndss.2019.23386

[51] Kevin K Lee, Sam Dutton, and Alex Turner. 2023. Shared storage API DEMO. https://shared-storage-demo.web.app.

[52] Janette Lehmann, Mounia Lalmas, Georges Dupret, and Ricardo Baeza-Yates. 2013. Online multitasking and user engagement. In *Proceedings of the 22nd ACM*

International Conference on Information & Knowledge Management (CIKM '13). Association for Computing Machinery, San Francisco, California, USA, 519–528. doi:10.1145/2505515.2505543

[53] Chao Liu, Ryen W. White, and Susan Dumais. 2010. Understanding web browsing behaviors through Weibull analysis of dwell time. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Geneva, Switzerland) *(SIGIR '10)*. ACM, New York, NY, USA, 379–386. doi:10.1145/1835449.1835513

[54] Minjun Long and David Evans. 2024. Evaluating Google's Protected Audience Protocol. *Proceedings on Privacy Enhancing Technologies* (2024).

[55] Marcos Cáceres. 2022. Shared Storage - Request for position on an emerging web specification. https://github.com/WebKit/standards-positions/issues/10.

[56] Martin Thomson. 2022. Negative position on shared storage. https://github.com/mozilla/standards-positions/commit/b6b62635c23000228f3061adeb18ed55da62a730.

[57] Jonathan R. Mayer and John C. Mitchell. 2012. Third-Party Web Tracking: Policy and Technology. In *IEEE Symposium on Security and Privacy (SP '12)*. IEEE, San Jose, California, USA, 413–427. doi:10.1109/SP.2012.47

[58] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012).

[59] Mozilla. 2023. Firefox rolls out Total Cookie Protection by default to more users worldwide. https://blog.mozilla.org/en/products/firefox/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/.

[60] Alexandra Nisenoff, Ranya Sharma, and Nick Feamster. 2023. User Awareness and Behaviors Concerning Encrypted DNS Settings in Web Browsers. In *USENIX Security Symposium (SSYM '23)*. USENIX, Anaheim, CA, 3117–3133. https://www.usenix.org/conference/usenixsecurity23/presentation/nisenoff-awareness

[61] Mike Perry. 2012. Do Not Beg: Moving Beyond DNT through Privacy by Design. https://www.w3.org/2012/dnt-ws/position-papers/21.pdf.

[62] Michiel Philipse, Güne Acar, and Christine Utz. 2024. Post-Third-Party Cookies: Analyzing Google's Protected Audience API. (2024). https://www.cs.ru.nl/masters-theses/2024/M_Philipse___Post-Third-Party_Cookies_Analyzing_Google's_Protected_Audience_API..pdf.

[63] Charles Reis, Alexander Moshchuk, and Nasko Oskov. 2019. Site Isolation: Process Separation for Web Sites within the Browser. In *28th USENIX Security Symposium*. USENIX, Santa Clara, CA, 1661–1678. https://www.usenix.org/conference/usenixsecurity19/presentation/reis

[64] Eduardo Schnadower Mustri, Idris Adjerid, and Alessandro Acquisti. 2023. Behavioral Advertising and Consumer Welfare: An Empirical Investigation. In *PrivacyCon*. https://www.ftc.gov/system/files/ftc_gov/pdf/PrivacyCon-2022-Acquisiti-Mustri-Behavioral-Advertising-Consumer-Welfare.pdf

[65] Asuman Senol and Gunes Acar. 2023. Unveiling the Impact of User-Agent Reduction and Client Hints: A Measurement Study. In *Proceedings of the 22nd Workshop on Privacy in the Electronic Society (WPES '23)*. Association for Computing Machinery, Copenhagen, Denmark, 91–106. doi:10.1145/3603216.3624965

[66] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. 2021. AdVeil: A Private Targeted Advertising Ecosystem. Cryptology ePrint Archive, Paper 2021/1032. https://eprint.iacr.org/2021/1032 https://eprint.iacr.org/2021/1032.

[67] Shivani Sharma. 2024. Fenced Frames Network side channel. https://github.com/WICG/fenced-frame/blob/master/explainer/network_side_channel.md.

[68] Peter Snyder. 2022. Privacy And Competition Concerns with Google's Privacy Sandbox. https://brave.com/web-standards-at-brave/6-privacy-sandbox-concerns/.

[69] Konstantinos Solomos, John Kristoff, Chris Kanich, and Jason Polakis. 2021. Tales of favicons and caches: Persistent tracking in modern browsers. In *Network and Distributed System Security Symposium*.

[70] Latanya Sweeney. 2013. Discrimination in online ad delivery. *Commun. ACM* 56, 5 (2013), 44–54.

[71] Martin Thomson. 2022. Privacy Preserving Attribution for Advertising. https://blog.mozilla.org/en/mozilla/privacy-preserving-attribution-for-advertising/.

[72] Martin Thomson. 2023. A Privacy Analysis of Google's Topics Proposal. https://mozilla.github.io/ppa-docs/topics.pdf.

[73] Martin Thomson. 2024. Protected Audience Privacy Analysis. https://mozilla.github.io/ppa-docs/protected-audience.pdf.

[74] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. 2010. Adnostic: Privacy preserving targeted advertising. In *Proceedings Network and Distributed System Symposium*.

[75] Florian Turati, Karel Kubicek, Carlos Cotrini, and David Basin. 2023. Locality-Sensitive Hashing Does Not Guarantee Privacy! Attacks on Google's FLoC and the MinHash Hierarchy System. *Proceedings on Privacy Enhancing Technologies* (2023).

[76] Alex Turner, Camilla Smith Barnes, Josh Karlin, and Yao Xiao. 2023. WICG/shared-storage: Explainer for proposed web platform shared storage API. https://github.com/WICG/shared-storage.

[77] Giridhari Venkatadri, Piotr Sapiezynski, Elissa M. Redmiles, Alan Mislove, Oana Goga, Michelle Mazurek, and Krishna P. Gummadi. 2019. Auditing Offline Data Brokers via Facebook's Advertising Platform. In *The World Wide Web Conference (WWW '19)*. ACM, San Francisco, CA, USA, 1920–1930. doi:10.1145/3308558.3313666

[78] Alberto Verna, Nikhil Jha, Martino Trevisan, and Marco Mellia. 2024. A First View of Topics API Usage in the Wild. In *Proceedings of the 20th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '24)*. Association for Computing Machinery, Los Angeles, CA, USA, 48–54. doi:10.1145/3680121.3697810

[79] Sophie Veys, Daniel Serrano, Madison Stamos, Margot Herman, Nathan Reitinger, Michelle L. Mazurek, and Blase Ur. 2021. Pursuing Usable and Useful Data Downloads Under GDPR/CCPA Access Rights via Co-Design. In *Symposium on Usable Privacy and Security (SOUPS '21)*. USENIX, Virtual Conference, 217–242. https://www.usenix.org/conference/soups2021/presentation/veys

[80] Miranda Wei, Madison Stamos, Sophie Veys, Nathan Reitinger, Justin Goodman, Margot Herman, Dorota Filipczuk, Ben Weinshel, Michelle L. Mazurek, and Blase Ur. 2020. What Twitter Knows: Characterizing Ad Targeting Practices, User Perceptions, and Ad Explanations Through Users' Own Twitter Data. In *29th USENIX Security Symposium*. USENIX, 145–162. https://www.usenix.org/conference/usenixsecurity20/presentation/wei

[81] WICG. 2024. Ad Selection API Proposal. https://github.com/WICG/privacy-preserving-ads.

[82] John Wilander. 2020. Full Third-Party Cookie Blocking and More. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/.

[83] Yuxi Wu, Sydney Bice, W. Keith Edwards, and Sauvik Das. 2023. The Slow Violence of Surveillance Capitalism: How Online Behavioral Advertising Harms People. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency (FAccT '23)*. Association for Computing Machinery, Chicago, IL, USA, 1826–1837. doi:10.1145/3593013.3594119

[84] Yuxi Wu, Panya Gupta, Miranda Wei, Yasemin Acar, Sascha Fahl, and Blase Ur. 2018. Your Secrets Are Safe: How Browsers' Explanations Impact Misconceptions About Private Browsing Mode. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 217–226. doi:10.1145/3178876.3186088

[85] Yingtai Xiao, Jian Du, Shikun Zhang, Wanrong Zhang, Qiang Yan, Danfeng Zhang, and Daniel Kifer. 2025. Click Without Compromise: Online Advertising Measurement via Per User Differential Privacy. arXiv:2406.02463 [cs.CR] https://arxiv.org/abs/2406.02463

[86] Haimo Zhang and Shengdong Zhao. 2011. Measuring web page revisitation in tabbed browsing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, Vancouver, BC, Canada, 1831–1834. doi:10.1145/1978942.1979207

[87] Ke Zhong, Yiping Ma, and Sebastian Angel. 2022. Ibex: Privacy-preserving Ad Conversion Tracking and Bidding. In *Proceedings of the 2022 ACM Conference on Computer and Communications Security (CCS '22)*. Association for Computing Machinery, Los Angeles, CA, USA, 3223–3237. doi:10.1145/3548606.3560651