Fledging Will Continue Until Privacy Improves: Empirical Analysis of Google's Privacy-Preserving Targeted Advertising

Giuseppe Calderonio, Mir Masood Ali, and Jason Polakis
University of Illinois Chicago
{gcalde22, mali92, polakis}@uic.edu

Abstract

Google recently announced plans to phase out third-party cookies and is currently in the process of rolling out the Chrome Privacy Sandbox, a collection of APIs and web standards that offer privacy-preserving alternatives to existing technologies, particularly for the digital advertising ecosystem. This includes FLEDGE, also referred to as the Protected Audience, which provides the necessary mechanisms for effectively conducting real-time bidding and ad auctions directly within users' browsers. FLEDGE is designed to eliminate the invasive data collection and pervasive tracking practices used for remarketing and targeted advertising. In this paper, we provide a study of the FLEDGE ecosystem both before and after its official deployment in Chrome. We find that even though multiple prominent ad platforms have entered the space, Google ran 99.8% of the auctions we observed, highlighting its dominant role. Subsequently, we provide the first in-depth empirical analysis of FLEDGE, and uncover a series of severe design and implementation flaws. We leverage those for conducting 12 novel attacks, including tracking, cross-site leakage, service disruption, and pollution attacks. While FLEDGE aims to enhance user privacy, our research demonstrates that it is currently exposing users to significant risks, and we outline mitigations for addressing the issues that we have uncovered. We have also responsibly disclosed our findings to Google so as to kickstart remediation efforts. We believe that our research highlights the dire need for more in-depth investigations of the entire Privacy Sandbox, due to the massive impact it will have on user privacy.

1 Introduction

The digital advertisement industry has grown to massive proportions since the first incarnations of web advertisement banners in the early 1990s, and is expected to surpass the trillion dollar mark within the next five years [1]. Online ads not only constitute a major revenue stream for smaller or independent content publishers and developers (e.g., mobile app developers), but are also the main revenue stream of tech giants like Google and Facebook. Ad strategies have also evolved over time, as advertisers aim to improve conversion through behavioral advertising, where users are targeted based on their demographics, preferences and interests. As a result, the ad ecosystem is powered by widespread data collection (typically performed by third-party tracking scripts) for generating user *profiles* based on their browsing habits. Personal user data is amassed by a limited number of companies that have a digital presence on a large number of websites [2, 3], leading to major privacy concerns due to the nature of the data [4–7] as well as the inferences they enable [8, 9].

This has resulted in the commodification of personal data [3], advertisers employing privacy-invasive practices, and ads enabling harmful or problematic behaviors [10–12]. At the same time, users have become increasingly concerned about their online privacy, leading to anti-tracking and ad-blocking capabilities that aim to curtail privacy-invasive behaviors being deployed by privacy-oriented browsers and tools [13, 14]. This has resulted in an arms race, where advertisers and trackers employ new techniques for bypassing anti-tracking defenses deployed by browsers and extensions [15–18].

Recently, Google announced their plans to completely phase out third-party cookies (which enable cross-site tracking) and deploy a series of systems that provide privacy-preserving alternatives to existing advertising mechanisms so as to allow advertisers to maintain their revenue stream while respecting users' privacy. Given Google's role and positioning within the web ecosystem, the Google Chrome Privacy Sandbox initiative may prove to be the most impactful development of recent years in the ad ecosystem, and various privacy organizations and browser vendors have voiced concerns about this further strengthening Google's monopoly and undermining users' privacy [19–21].

In this paper, we present the first comprehensive exploration of the FLEDGE API,[1] which provides the mechanisms and infrastructure needed for running real-time ad auctions without compromising user privacy [22, 23]. The main idea behind

[1] While FLEDGE was recently renamed as the Protected Audience API, we will refer to it as FLEDGE for conciseness.

FLEDGE is to directly execute auctions in the browser, instead of running them on remote servers. By shifting computation to the client side, the need for sharing personal data with third parties can be eliminated. To accomplish that, Chrome has deployed a new type of browser storage called Interest Groups, which can be used to store the information about user preferences that is necessary for successfully running ad auctions. Initially, we provide a series of snapshots capturing the operational status of the FLEDGE API over a period of four months, capturing advertisers experimentation with the system prior to its release as well as after its official deployment in Chrome. Our study reveals a large number of services that have incorporated FLEDGE into their workflow, as well as Google's dominant role within the ecosystem. Subsequently, we conduct an in-depth investigation and empirical analysis of FLEDGE, culminating in twelve novel attacks. We present a series of techniques that target design or implementation flaws in FLEDGE's operation and enable four different classes of attacks: (i) tracking, (ii) cross-site information leakage, (iii) service disruption, and (iv) pollution attacks. In a nutshell, we demonstrate how websites can misuse FLEDGE to re-identify and track users across domains without any cost, partially infer a user's browsing history, prevent other sites from showing the user ads, crash the user's browser, and pollute a user's interest groups so as to manipulate the ads that they will ultimately be shown.

Due to the severe implications of our findings we have disclosed our research to Google. While they patched the issue that allowed us to crash a user's browser, our other attacks still affect Chrome's official deployment of FLEDGE. While some of our attacks may be mitigated by pending changes that Google has announced (e.g., deploying a private aggregation server), others will not be affected. Overall, we strongly believe that additional scrutiny of all Privacy Sandbox proposals is necessary for ensuring that users' privacy is not irreversibly undermined by Google's ongoing efforts.

In summary, our work presents the following contributions:

- We present a large-scale longitudinal measurement of the emerging FLEDGE ecosystem, shedding light on its inner workings and key market shareholders.
- We present the first, to our knowledge, comprehensive empirical analysis of Google's FLEDGE API, and demonstrate novel attacks with severe security and privacy implications.
- We have disclosed our findings to Google, and also propose a series of mitigations against our attacks. We have released our code and data to facilitate additional research [24].

2 Background

The advertising ecosystem plays a pivotal role in facilitating the interaction between publishers and advertisers, ensuring the efficient sale and purchase of ad space. This section provides an overview of the current advertising landscape, focusing on the underlying technologies and platforms involved in the process, and how they will be influenced by the introduction of the FLEDGE API. To understand the process of selecting relevant ads for users, it is essential to first briefly introduce the key parties involved.

Publishers are websites seeking to monetize their platform by offering ad space to advertisers, and **advertisers** are websites aiming to promote their products or services through ads.

Ad Networks. To streamline the complex process of managing delivering ads, various intermediaries that are collectively referred to as ad networks have emerged. These serve as distributed systems that connect publishers and advertisers, and are comprised of several components:

DSPs are services for advertisers to automate the purchasing of ad impressions across multiple publisher sites.

SSPs automate the selling of ad inventory for publishers, enabling them to offer their available ad space to various ad exchanges, DSPs, and networks.

Ad Exchanges function as platforms that automate the buying and selling of ad inventory from multiple ad networks. They act as intermediaries between SSPs and DSPs.

DMPs are data management platforms that store user information. Websites subscribe to DMP platforms to benefit from centralized data organization, while DMPs gather and sell user data, e.g., for real-time bidding.

Real-time bidding (RTB) is a process that aims to show users relevant ads that are tailored to their actual interests, which will be displayed on an publisher's page from a pool of available advertisers. In a nutshell, as outlined in Figure 1, this operates as follows: when a user's browser sends a GET request to a publisher's website, the returned page includes an ad slot. For simplicity, we assume a single ad slot, although multiple slots can exist in practice. The publisher embeds the URL of the SSP to retrieve the ad within the ad slot, and this URL contains crucial bidding-related data, such as the user ID and publisher URL. Upon receiving the request, the SSP processes it to comply with the ad exchange API standards and forwards it to the ad exchange server, which then routes the request to the most suitable DSP based on user preferences and ad type. The DSP will generate a bid request using the user ID and received data, subsequently returning the bid and associated ad URL to the ad exchange. The ad exchange will compute the winning bid and provide the SSP with the corresponding ad URL, and the SSP will deliver the ad URL back to the user's browser, which will fetch and display the ad. Even though this process involves multiple steps, it typically occurs within an short amount of time (e.g., average of 300ms for 24 bidders [25]). Additionally, DMPs and DSPs/SSPs exchange information about user preferences asynchronously with respect to the real-time bidding process. The specifics of this exchange may vary based on individual agreements, and are commonly implemented using APIs exposed by DMPs. DSPs and SSPs are responsible for storing relevant information regarding user preferences asynchronously, thereby enhancing the targeting capabilities of the advertising ecosystem.

Privacy concerns. The effective operation of the advertising process relies on the user IDs, which are typically obtained through third-party cookies, and stored within DMPs that employ various tracking techniques and JavaScript APIs to collect data and create comprehensive user profiles, each associated with a unique user identifier. However, the reliance on third-party cookies raises significant privacy concerns. Users often lack awareness and control over the sharing of their information, including the transmission of sensitive data exploited by DMPs to create detailed and accurate user profiles.

2.1 Privacy Sandbox Proposals

Google's Privacy Sandbox initiative constitutes a concerted effort to gradually phase-out third-party cookie support from the Chrome browser while offering privacy-preserving systems that will fill that void, thus allowing the advertising ecosystem to continue to operate in an effective manner.

FLEDGE (First Locally-Executed Decision over Groups Experiment) focuses on interest-based advertising. It introduces the novel `interestgroups` browser storage and provides a comprehensive set of browser API methods. These methods facilitate the creation and management of interest groups within the user's browser, allowing interest-based ad auctions to occur locally while preserving user privacy.

The core principle behind FLEDGE is to securely store user information within the browser itself, avoiding the transmission of data over the internet. This approach enables real-time bidding to take place within the browser, bypassing the need for ad exchanges. Importantly, FLEDGE's browser API can be leveraged by both publishers and advertisers, granting the flexibility to delegate ad management to SSPs and DSPs while maintaining privacy-enhancing measures.

It is essential to note that during the majority of the time in which this study was conducted, the FLEDGE API remained an experimental feature. This was a deliberate decision on our part, as our goal was to conduct a security and privacy analysis of FLEDGE in an effort to allow Google to address any flaws uncovered by our investigation prior to FLEDGE being officially released and potentially affecting billions of users. Informally, FLEDGE became available for testing purposes as of version 101.0.4951.26 in April 2022, and was officially released to the public and integrated into the stable Chrome browser as of version 117.0.5938.89 in mid September 2023. Despite being officially released, the current implementation still relies on mechanisms that Google has labeled as "temporary", with the plan of substituting them once third-party cookie deprecation is completed.

Interest Groups. The FLEDGE API introduces the concept of interest groups as a cross-context information storage mechanism (replacing third-party cookies) specifically intended for holding information for in-browser ad auctions. Essentially, interest groups contain advertisements and specific information about user interests based on their browsing activity. Each interest group is associated with a specific owner (the origin of the advertiser or DSP), and contains a set of fields, including the name of the group, the bidding logic (links to JavaScript code used to score the bid value), along with ads (renderUrl) and ad components. Unlike third-party cookies which are limited to 180 per-site in Chrome [26], interest groups are comprised of large JSON objects with a current limit of 2,000 interest groups (previously 1,000) associated with a single origin, and have a 10MB size limit. Interest groups are stored in the browser and are initially populated by the owner (advertiser or DSP) that created it. Thereafter, the browser refers to the `updateURL` field to make daily fetch requests to the advertiser's server for new versions. All interest group fields can be overwritten by an update, except for the name and owner.

Workflow. The functionality needed by FLEDGE is mainly implemented through four browser API calls.

`navigator.joinAdInterestGroup()`: This adds a new entry to the browser's storage in JSON format, with multiple fields and a specified lifetime (up to 30 days). It allows advertisers and DSPs to associate interest groups with the browser.

`navigator.leaveAdInterestGroup()`: By providing the relevant JSON format, this call removes an existing entry from the browser's storage. Advertisers and DSPs can utilize this call to remove interest groups when necessary.

`navigator.updateAdInterestGroups()`: This call updates the fields of all interest groups associated with the interest group owner in the user's browser. The updates are fetched from a trusted server, enabling advertisers and DSPs to keep the interest group information up-to-date.

`navigator.runAdAuction()`: This call initiates an auction process to determine the best ad, based on the auction input data provided as a JSON object. Publishers and SSPs use this call to run the auction and select the winning ad. Differently from the other API calls, this function returns the winning ad as an `opaqueURL`, a URL which hides the origin and path, while maintaining the website reference it points to.

FLEDGE's workflow can be summarized in two phases; first, when a user visits advertiser pages, either the page or a DSP on its behalf stores an interest group in the browser using `joinAdInterestGroup()`. The stored interest group contains information such as the owner (origin of the advertiser or DSP), the name of the group, its bidding logic (URL with code used by the advertiser/DSP to score the bid value), and a set of ads (URLs). Second, when the user visits a publisher's page where an ad needs to be displayed, the publisher or an SSP runs an auction using `runAdAuction()`. The auction specifies the seller (origin of the auction runner), the interest group buyers (an array of URLs representing the advertisers/owners), and the decision logic (URL with code used by the publisher/SSP to determine the winning bid and the ad to be displayed) in its most basic form. If an advertiser owns at least one interest group in the browser, it participates in the auction with its associated groups.

During the auction conducted by the publisher or SSP, `biddingLogicUrl` and `decisionLogicUrl` play a crucial

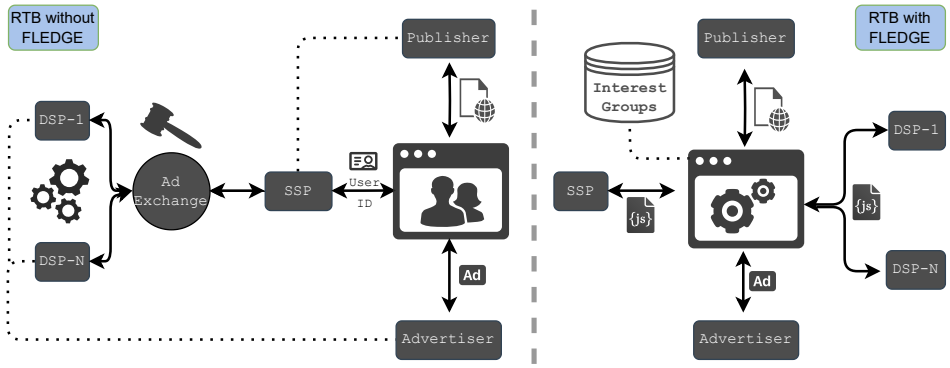

Figure 1: Overview of the traditional RTB workflow (left) and how that changes in the presence of FLEDGE (right).

role in determining the winning bid and the ad to be displayed. These URLs expose a JavaScript file containing the `generateBid()` and `scoreAd()` functions respectively. When the auction takes place, the two URLs are fetched and the functions declared inside are invoked in turn, for each interest group participating in the auction. `generateBid()` calculates the bid value based on the provided inputs and returns the bid value and the URL of the ad to render if the bid is successful. `scoreAd()` evaluates the bid value and additional metadata for each interest group. By considering this information, it generates a desirability score that helps determine the winning bid and the ad to display; in practice, the interest group with the highest desirability score wins the auction.

Within the `decisionLogicUrl` Javascript file, interest group buyers can optionally include the `reportWin` function. This serves the purpose of managing reporting activities for the winning interest group. It is important to note that the future implementation of this functionality will be integrated with the Private Aggregation API [27], according to Google's reports. However, as a temporary mechanism, a `sendReportTo` function allows sending a single GET request to the owner website and include reporting data as query parameters.

Additionally, other resources/URLs can be specified both by advertisers/DSPs and publishers/SSPs when joining interest groups or running an auction, which will be fetched at auction time. In particular, `trustedBiddingSignalsURL` for buyers and `trustedScoringSignalsURL` for sellers allow the designated party to fetch bidding-related data from a *Trusted Server*, intended to preserve user's privacy and avoid logging sensitive information.

After all interest groups have made their decision and the seller has evaluated them, the auction completes by returning the opaque URL associated with the winning ad to the JavaScript file. Upon completion, the seller should manually render the ad within a `fencedframe`, a modified version of an iframe which accepts an `opaqueUrl` and does not allow network access or visibility for the top page context. In future implementations of FLEDGE, the ad rendering mechanisms will be conducted with pre-rendering mechanisms such as web bundles. However, until third-party cookies are deprecated, ads will be fetched from the advertiser's website.

Finally, Chrome is planning to establish constraints regarding the ads that can be rendered, wherein at least 50 different browsers should have joined an interest group with the same ad, name and `biddingLogicUrl` for it to be displayed; essentially, this constraint will implement a form of `k-anonymity`.

Trusted Server. In addition to the information stored within interest groups, both buyers (via `trustedBiddingSignalsURL` in interest groups) and sellers (via `trustedScoringSignalsURL` in auction config) can fetch real-time signals from their own servers to be used in the bidding and scoring logic during auctions. Since this information is sensitive and could be used to track users, FLEDGE intends to enforce restrictions on the design of these servers. Specifically, the server must be a first-party server, must solely provide a key-value service, must not log incoming requests, and must handle the key-value service within a Trusted Execution Environment (TEE). However, the enforcement and trust-model for such servers has not yet been finalized or enforced, even after FLEDGE's official public release. Instead, a Bring Your Own Server (BYOS) "temporary" solution has been made available, allowing buyers and sellers to send themselves the URLs instead of relying on a trusted server, and fetching data directly from their non-isolated servers. As a result, the existing mechanism allows for the potential of tracking users and leaking sensitive information, and is not in line with the privacy-preserving goals of FLEDGE. Finally, we note that even TEEs have been shown to be vulnerable to attacks [28–32].

K-anonymity. To ensure that unique interest groups are not used to track individual users, Chrome is *planning* to establish constraints regarding the ads that can be rendered, wherein at least 50 different browser instances should have joined an interest group. If interest groups do not satisfy this constraint, they will not be made available during auctions. However, the k-anonymity constraint does not apply to the entire interest group and is instead enforced on specific fields, i.e., name, `biddingLogicUrl`, and ad (including `renderUrl`). Additionally, the 50-browser limit will be determined based

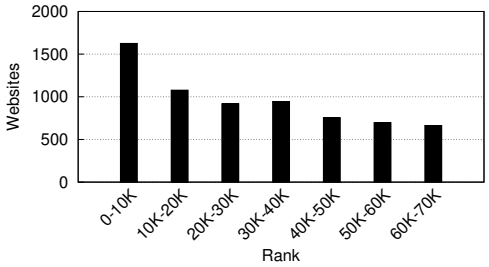

Figure 2: Publishers using FLEDGE, ranked by popularity.

on requests received from hidden browser IPs (routed through Oblivious HTTP [33]). We further discuss this in §5.

Privacy guarantees. FLEDGE aims to offer a privacy-preserving solution by shifting computation and bidding processes to the user's browser, eliminating the need for data sharing through third-party cookies. Users can gain better control over their data with interest groups managed directly in the browser. In their original framework explainer, the developers highlighted three key *privacy advancements* that the proposal is designed to achieve [34]:

- **PA1:** The browser, not the advertiser, holds the information about what the advertiser thinks a person is interested in.
- **PA2:** Advertisers can serve ads based on an interest, but cannot combine that interest with other information about the person — in particular, with who they are or what page they are visiting.
- **PA3:** The websites that the person visits, and the ad networks those sites use, cannot learn about their visitors' ad interests.

While FLEDGE aims to provide a privacy-conscious approach to targeted advertising, an in-depth investigation is necessary for assessing whether it meets those privacy advancements. As we show in §4, the current design and implementation of FLEDGE are vulnerable to numerous attacks that violate its fundamental privacy goals, exposing users to significant risk.

3 Measurements

In this section, we provide the first large-scale measurement of FLEDGE being deployed across the web ecosystem.

Methodology. To automate the process of identifying websites that use the FLEDGE API, we used Puppeteer to orchestrate Chrome. We visit each domain using a fresh browser instance to ensure that we correctly attribute our observations to individual publishers. Once each homepage is fully loaded, our crawler parses the DOM to gather a list of up-to twenty first-party links to visit, to increase coverage and the possibility of observing and capturing the use of relevant functionality. We detect the use of the FLEDGE API by developing function wrappers for calls made to `navigator.runAdAuction`, `navigator.joinAdInterestGroup`, and `navigator.leaveAdInterestGroup`. The Puppeteer script injects wrapper functions into the page and logs all corresponding accesses. After visiting all links under a domain, we close the browser and collect the SQLite Database where Chrome stores information about the Interest Groups of each browsing profile. Finally, we clear the state and all relevant files associated with the browsing session.

Table 1: Overview of sellers over time.

Month	Seller	#Publishers	#Auctions
June	securepubads.g.doubleclick.net	1,761	5,738
July	securepubads.g.doubleclick.net	1,038	2,960
	cdn.mediago.io	1	1
September	securepubads.g.doubleclick.net	565	3,243
	cdn.mediago.io	9	17

Dataset. Using our automated approach, we perform a crawl of the top 70K domains from the Tranco list [35]. We opted for this set based on the hypothesis that popular domains are more likely to expeditiously develop the infrastructure necessary for using new features introduced by the Privacy Sandbox. Our framework collected the API calls on these sites in June and July 2023, thereby capturing snapshots of FLEDGE usage while it was an experimental feature, and again in September 2023 when it was publicly released.

Publishers. We detected the use of FLEDGE APIs in 9.5% ($n = 6{,}682$) of the 70K sites that we visited during the measurement period. Figure 2 presents an overview of the sites ranked by popularity. Indeed, we observe a correlation between websites' popularity and the usage of FLEDGE, as it was most prevalent in domains from the top 10K. Interestingly, over the three measurement snapshots we observe a decline in the number of sites using FLEDGE. While 4,628 sites used FLEDGE in June, this number decreased to 3,903 sites in July, and further dropped to 3,585 sites in September. Since our measurements also covered a transitionary period for FLEDGE, wherein it evolved from a set of experimental features to a public release, the reduction in the number of sites using the system could indicate that sites are still experimenting with how to fully and effectively take advantage of FLEDGE features prior to its official release. We expect the number of sites to significantly increase as the system matures and grows more stable, with advertisers and sellers scaling support and updating their infrastructure. Importantly, while we captured FLEDGE calls within multiple top-level publisher domains, all the actions we observed were performed by third-party advertisers and sellers. Our observations indicate that Privacy Sandbox proposals may likely end up primarily being adopted by larger, ecosystem-wide entities instead of benefiting smaller, independent domains intending to remarket their products to their visitors and customers.

Sellers. We observed a total of 11,959 auctions for ad spaces during our measurement period, almost all of which ($n = 11{,}941$) were run by Google Ad Manager. Table 1 shows the number of auctions and the corresponding publishers

Table 2: Interest groups' `join` and `leave` actions.

Owner	#Publishers	Interest Groups #Joined	Interest Groups #Left
td.doubleclick.net	2,924	12,190	2,533
fledge.as.criteo.com	370	880	-
fledge.teads.tv	229	1	-
fledge.eu.criteo.com	327	918	-
fledge.us.criteo.com	655	1411	-
fledge-eu.creativecdn.com	95	261	52
fledge-usa.creativecdn.com	65	133	37
fledge-asia.creativecdn.com	56	129	17
f.creativecdn.com	9	9	-
googleads.g.doubleclick.net	5	5	-
cdn.mediago.io	5	1	-
at-us-east.amazon-adsystem.com	1	1	-
adthrive.com	4	4	-

within which they were ran. Our findings about the dominance of Google Ad Manager are in-line with prior observations of the ad ecosystem [36]. Additionally, since the Privacy Sandbox initiatives are led by Google and are geared towards Chrome browsers, it is expected that their ad ecosystem will integrate the latest functionality into their operations. Finally, Google Ad Manager has been running FLEDGE operations on a small set of publishers, planned to scale to 10% of auctions by the end of 2023 [37, 38]. While most auctions that we observed were run by Google, these auctions also included a large number of *component auctions*, i.e., separate auctions from multiple sellers, the results of which were passed to the top-level auction. We report on these auctions later in this section. Nonetheless, other popular sellers including Xandr, Teads, RTBHouse, OpenX, PubMatic, and Criteo [39–42] have been working on integrating FLEDGE auctions into their workflows. We expect that future measurements of the ecosystem will include additional sellers once the ecosystem matures further.

Advertisers. We observed 7 advertisers (that map to 13 different domains) adding users to interest groups and participating in FLEDGE auctions. Table 2 provides an overview of the advertisers that we observed during our measurements. We observed three large advertising companies, Google (DoubleClick), Criteo, and RTB House (CreativeCDN), using the APIs from multiple domains. Since FLEDGE actions are restricted based on origin (i.e., protocol, domain, and port), using multiple origins enables advertisers to act as multiple individual participants in creating interest groups and placing bids. While this approach restricts interest group access to individual origins, advertisers can expand the size of their interest group set and bypass the current restriction of 2,000 groups per-origin [43]. Additionally, they can access multiple origin-based dedicated worklets to run bidding logic and also place multiple bids in a single auction [44].

Since FLEDGE allows advertisers to use personal data for ad personalization and access storage through interest groups, the EU User Consent Policy requires advertisers to gather user

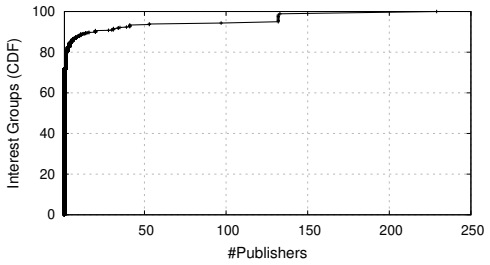

Figure 3: Publishers associated with each interest group.

consent before performing such actions [45]. Interestingly, we observed Criteo and RTB House creating region-based subdomains and adding browsers to multiple interest group sets under subdomains dedicated to EU, US, and Asia-based users. While all the measurements were performed through US-based IP addresses, we observed the browser joining interest groups from other regions as well. Advertisers may choose to create multiple interest group sets in an attempt to comply with varying consent and data handling regulations.

Interest Groups. We observed a total of 15,935 interest groups; Figure 3 shows the number of publishers within which the browser was added to each interest group. We found that 92% of the observed interest groups were only added upon a visit to a single top-level domain. Our observation is in-line with the intended use of interest groups, i.e., storing user interests based on their browsing activity. Additionally, due to the granular nature of this information it is crucial that the data held within these groups is protected, preventing leaks and misuse for privacy-invasive user tracking.

FLEDGE caps interest groups lifetime to 30 days, but it can be extended by repeating a call to `navigator.joinAdInterestGroup()` [44]. We observed 18.28% ($n = 2,913$) of the interest groups across all three dataset snapshots. Interestingly, while we observed 306,407 calls to join an interest group, 30.90% ($n = 94,673$) attempted to create an interest group with an expiration time greater than 30 days. While the browser enforces the 30-day expiration limit of interest groups, this indicates advertisers' attempts at making interest groups persist longer within browsers, which also explains the limited removal of interest groups shown in Table 2.

While joining and updating groups, advertisers can include arbitrary information within bidding signals that help inform bidding logic when the interest group is included in an auction. They can specify a list of keys in the `trustedBiddingSignalsKeys` field, for which values will be fetched in real-time from a key-value service that the advertiser hosts. We observed limited use of this approach with advertisers adding only 1 key to an interest group on average. Additionally, they may also include information through key-value pairs within the `userBiddingSignals` field. The values stored in this field will remain in the user's browser and be made available to the advertiser within on-device auctions. Given that advertisers

Table 3: Summary of our attacks, the mechanism they misuse, privacy advancements they violate, and planned mitigations.

Type	Mechanism	Field	Attacker Role	Future Mitigation	Violation
Tracking	Bidding Helpers	biddingWasmHelperUrl	Advertiser & Seller	Not Planned	PA2
Tracking	Real-time Rendering of Winning Bids	biddingLogicUrl	Advertiser & Seller	Fenced Frames	PA2
Tracking	Bidding Logic	ads	Advertiser & Seller	Not Planned	PA2, PA3
Tracking	Trusted Bidding Signals	trustedBiddingSignals	Advertiser & Seller	Trusted Server	PA1, PA2
Tracking	Non-aggregated Win Reporting	reportWin	Advertiser & Seller	Private Aggregation	PA2, PA3
Tracking	Non-aggregated Win Reporting	sendReportTo	Advertiser & Seller	Private Aggregation	PA2
Tracking	Non-aggregated Event-level Reporting	reportEvent	Advertiser & Seller	Fenced Frames	PA2, PA3
Cross-site Leak	Gathering Interest Group Owners	decisionLogicUrl	Advertiser (Run Auctions)	Not Planned	PA2, PA3
Cross-site Leak	Interest Group Leaks	trustedScoringSignal	Advertiser (Run Auctions)	Trusted Server	PA1, PA2, PA3
DoS	Browser Crash	trustedBiddingSignals	Advertiser & Seller	Fixed	Other
DoS	Blocking Ad Auctions	interestGroups.sqlite3	Advertiser (Join Groups)	Not Planned	Other
Pollution	Polluting Doubleclick Interest Groups	Interest Group size limit	Other (Add iframes)	Not Planned	Other

do not need to set up a server-side service to `userBiddingSignals`, we found that they used the field in 92.59% ($n = 283{,}714$) to join interest groups and included 3 key-value pairs on average within an interest group ($SD = 0.59$).

Each interest group can store and bid on multiple ads. We observed that advertisers populated interest groups with 18 ads on average ($SD = 32$, $M = 7$). Additionally, ads can be further supplemented by ad components, intended to hold pieces (e.g., individual products) that can be associated with ad campaigns. While we found that only 1.4% ($n = 4{,}306$) of calls to join an interest group populated the `adComponents` field, they stored extensive information, populating it with 132 components on average ($SD = 96.5$, $M = 100$), Our observations highlight that interest groups are complex objects that contain data in ways that could not be previously captured within third-party cookies. With an extensive limit of 2,000 groups per-origin, coupled with a lax k-anonymity requirement on this information, our findings further motivate the importance of an in-depth analysis of FLEDGE's privacy implications.

Auctions. We observed a total of 11,959 auctions, wherein we observed 4 auctions on average ($SD = 6.4$,$M = 2$) for each publisher across 3 visits. FLEDGE includes support for component auctions, and their results are passed to the top-level auction. Out of 11,959 top-level auctions, we observed a total of 11,994 component auctions ($Avg. = 1$, $SD = 0.6$, $M = 1$). These included a limited number of participants, with 2.14 advertisers on average ($SD = 1.49$, $M = 2$). Moreover, sellers can include arbitrary information for assisting them in scoring bids. We found sellers making extensive use of this field, passing 20.88 signals on average ($SD = 3.95$, $M = 21$) for use in scoring bids gathered within component auctions. Additionally, FLEDGE allows sellers to pass arbitrary signals to individual buyers by using the `perBuyerSignals` field. We observed sellers passing extensive information to each buyer within individual component auctions, with seller passing 9.96 signals on average ($SD = 6.28$, $M = 6$) to each buyer. Since the values within these signals can be arbitrary, we observed that all signals were either passed as an array or an object. While the values contained were themselves obfuscated, the data structures were extensive, including additional arrays and objects, resulting in an average depth of 7.6 ($SD = 4.27$, $M = 6$).

Summary. Overall, we observed that FLEDGE interest groups and auctions include a large number of components that store granular user information and share arbitrary objects between multiple third-party entities. The inclusion of on-device logic execution, coupled with fetching real-time data from servers, and support for multiple sub-auctions creates a large ecosystem that needs to ensure proper handling of data, especially within fields that can hold arbitrary values. Our measurements highlight the adoption of the system across multiple publishers and its on-going use by prominent advertisers. Since Google has enabled the system by default for Chrome users, an empirical analysis of FLEDGE is necessary for understanding the inner workings of the ecosystem and to evaluate the system's effectiveness in preventing privacy-invasive misuse.

4 Empirical Analysis of FLEDGE

In this section, we present a series of novel attacks that misuse FLEDGE, and detail the functionality that we leverage to achieve them. Table 3 provides an overview of our attacks and highlights the privacy goals that each attack violates.

Threat model. Our attacks are performed by a web attacker that uses the FLEDGE API as an advertiser and/or seller. We assume that the user visits a website where the attacker has the ability to add the browser to interest groups, and to participate as a seller by running auctions. To that end, the attacker needs to own a domain with the server having been setup to handle FLEDGE requests, including support for key/value services and file responses for individual fields used by interest groups and auctions. Beyond the infrastructure setup, the threat model assumes a weak attacker with limited privileges since our attacks can be carried out by attackers embedded as third-party resources, and do not require the user to visit the attacker's domain. As FLEDGE APIs are enabled by default in Chrome, users do not need to enable any experimental flags.

Experimental setup and auditing process. The attacks we present in this section were designed and performed using

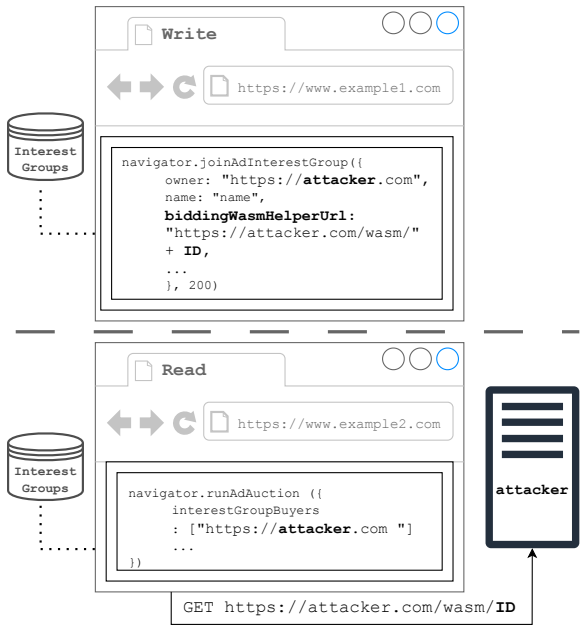

Figure 4: User tracking with bidding helpers.

our own infrastructure, and were guided by our detailed study of the FLEDGE documentation, code review, and empirical experimentation. This included creating test pages that use the FLEDGE APIs, understanding the purpose of each parameter, analyzing the flow of information across different entities, and identifying potential privacy violations and attack vectors. We then created custom exploits using controlled web pages to evaluate the attacks' practicality and feasibility. We did not perform any attacks on live users or websites. We used our own domains, hosted at an institutional IP address, and accessed these domains using Chromium browsers on our own machines. For the attacks described in §4.1 and §4.2, the scripts simulated both victim and attacker scenarios. Finally, for the attack describing pollution of Doubleclick interest groups, we made calls to Doubleclick from a website under our control and within browsing sessions on our own devices.

4.1 User Tracking

Bidding Helpers. FLEDGE allows advertisers to populate interest groups with multiple fields that describe the name and contents of the group but also determine how the group will participate in ad auctions. Advertisers can configure custom bidding logic using JavaScript through the `biddingLogicUrl`. In addition to the JavaScript code, advertisers can execute computationally-expensive subroutines by directing the browser to fetch WebAssembly code from a link provided under the `biddingWasmHelperUrl`. The API enforces restrictions on this field; first, the provided link should return a WebAsssembly binary with an `application/wasm` MIME type, and the browser will make the returned code available while generating bids. Second, the link must share its `origin` (i.e., protocol, domain, and port) with that of the owner of the interest group. However, the API does not enforce k-anonymity restrictions on this field. Therefore, each instance of the interest group can set a unique `biddingWasmHelperUrl` for identifying a specific browser. Figure 4 presents an overview of this attack.

Writing an identifier. An attacker can add the browser to a specific group by passing the interest group object as an argument to `navigator.joinAdInterestGroup()`. The attacker can embed a unique 32-bit identifier corresponding to a specific browser instance into the path of the `biddingWasmHelperUrl` field of the interest group object.

Reading an identifier. In subsequent browsing sessions, the attacker can participate in auctions as both a seller and a buyer. As a seller, the attacker can create a unique, single-buyer auction with its own origin as the only participant. Upon running the auction, the browser will trigger a fetch request to the `biddingWasmHelperUrl`, thereby passing the identifier to the attacker's server, enabling it to re-identify the browser instance.

Mitigation. This attack leverages two aspects of `biddingWasmHelperUrl`'s functionality. First, the API does not impose anonymity restrictions on the uniqueness of the link for each browser instance. Second, the logic is fetched through a network request at runtime. In order to restrict the field from being misused, the FLEDGE API should first extend its enforcement of k-anonymity restrictions to include `biddingWasmHelperUrl` so as to prevent unique identifiers from being set for each browser. Furthermore, the browser should trigger a single fetch for WebAssembly code upon joining an interest group, which should be cached and reused for all future auctions involving that specific interest group.

Real-time Rendering of Winning Bids. When an interest group wins an auction, the browser refers to the link specified in the interest group's `renderUrl` field, gathers the necessary components, and renders the ad within an embedded frame. As part of the privacy protections introduced by the Privacy Sandbox proposals, Google is implementing a new HTML Element called Fenced Frame. This element is similar in design to an iframe, but is restricted in its ability to interact with other frames in the page (including the top-level frame) and also has restrictions to network access. The FLEDGE API proposes rendering winning ads within fenced frames. This way, the winning bid is rendered in a restricted environment with limited ability to perform privacy-invasive actions while embedded in a third-party site. Moreover, given the restrictions on network requests within fenced frames, the browser will fetch ads from the `renderUrl` in an asynchronous manner, independent of the execution of specific auctions. However, the Fenced Frame proposal is experimental, and its use remains optional in the current public release of the FLEDGE API, which allows winning bids to trigger real-time network requests to the `renderUrl`. Figure 5 presents an overview of this attack.

Writing an identifier. An attacker can create and maintain 32 unique strings, each mapping to one bit of a 32-bit

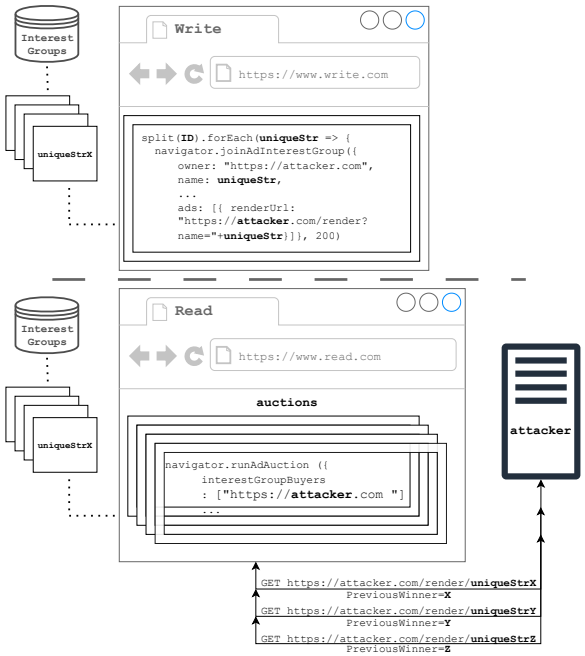

Figure 5: User tracking with rendering of winning bids.

identifier. For each new user the attacker gathers the list of strings that correspond to 1-valued bits and embeds them within the `name` and `renderUrl` fields of different interest groups. All interest groups share the same `owner` domain. The attacker makes the browser join all created interest groups by calling `navigator.joinAdInterestGroup()`. FLEDGE plans to impose a k-anonymity constraint on the `name`, `biddingLogicUrl`, and `renderUrl` fields. Thus, we avoid embedding the entire ID in a single interest group, and instead split it across 32-unique strings. Doing so reduces the number of unique interest groups that are required to satisfy the k-anonymity constraint. Additionally, in the presence of k-anonymity restrictions on other fields, the attacker ensures that all interest groups use the same `biddingLogicUrl`.

Reading an identifier. The attacker runs multiple single-buyer auctions, one for each stored interest group. With each auction, the browser calls the bidding logic and passes seller-defined `auctionSignals` as arguments. Since these can include arbitrary information, we add the list of the winners of completed auctions. The code within `biddingLogicUrl` can choose a random interest group to win each auction after filtering out the previous winners. This way, each group wins exactly once. Upon the completion of each auction, the browser triggers a network request to the `renderUrl`, and informs the attacker about the winner. The attacker can maintain a list of previous winners and use them for future auctions until all bits that correspond to a value of 1 have been gathered and a unique identifier has been reconstructed.

Mitigation: This attack is currently feasible due to the public release of the FLEDGE API before the development of privacy protections offered by Fenced Frames. Fetching the browser ads asynchronously will ensure that this attack cannot synchronously gather and maintain a list of winners and, thus, cannot regenerate an identifier across multiple auctions.

Bidding Logic. We also found a similar attack that makes use of the `biddingLogicUrl` instead of the `renderUrl`. The `biddingLogicUrl` contains JavaScript code that is invoked when the interest group is made part of an auction. As part of the FLEDGE API plans for the introduction of a k-anonymity server, advertisers will be required to ensure that the same `biddingLogicUrl` is present in at least 49 other browser instances. Even though the attack we present can be executed in spite of the anonymization requirement, the current public release of FLEDGE does not enforce any such restrictions.

Writing an identifier. The attacker again generates 32 unique strings, and gathers strings mapping to a bit value of 1. The attacker then creates interest group objects that share the same owner domain, and the corresponding string for each gathered bit is embedded in the `name` and `biddingLogicUrl` fields. The attacker will then make the browser join all created interest groups. When FLEDGE actually enforces k-anonymity in practice, the attacker would be required to ensure that interest groups corresponding to each unique string are available on at least 49 other browser instances (we discuss this and potential bypasses in §5). To do this, the attacker can design 32 unique interest groups, each corresponding to one bit of a 32-bit identifier. The attacker must ensure that interest groups containing a `biddingLogicUrl` corresponding to either value satisfies the anonymity requirement.

Reading an identifier. The attacker runs one single-buyer auction and the browser calls the `biddingLogicUrl`s corresponding to each interest group stored in the browser. The incoming requests let the attacker's server learn the value for each bit corresponding to a value of 1 that is stored in the browser. The attacker can fill in the values of 0 for the bits that are not stored in the browser and reconstruct the user's identifier. This attack is similar to the one presented in Figure 5, but the attacker reads the identifier during the execution of the auction, rather than when it wins the auction.

Mitigation. While introducing the k-anonymity constraint will impose restrictions on this attack, it does not eliminate the possibility of re-identifying the user. For this this attack to be effectively mitigated, the browser needs to fetch and cache the `biddingLogicUrl` in an asynchronous manner instead of gathering the code when running an auction. That way, the attacker receives requests for the `biddingLogicUrl` at arbitrary times and cannot map the request back to an individual instance or browsing session to reconstruct the identifier.

Trusted Bidding Signals. The FLEDGE API proposal aims to incorporate trusted servers into its auction infrastructure. These trusted servers can be used to add real-time bidding signals to bidding logic, which will execute within a Trusted Execution Environment (TEE). As this infrastructure is under development, Chrome allows advertisers and sellers to use cus-

tom implementations through their support for a *Bring Your Own Server* (BYOS) model. While this is a temporary solution to help with eventual migration to a TEE infrastructure, the BYOS model was made available by default in the public release of the FLEDGE API. To support the BYOS model, when running an auction the browser automatically passes the names of all stored interest groups that share the same key/value service defined by the `trustedBiddingSignalsUrl` field. An attacker can currently exploit this mechanism to embed and regenerate a unique identifier to track users across different sites.

Writing an identifier. The attacker generates the identifier and creates the unique interest group objects for each bit that corresponds to a value of 1. All groups share the same `owner` domain and the same `trustedBiddingSignalsUrl`, and the corresponding string for each bit is embedded in the `name` of the interest group fields. The attacker makes the browser join all created interest groups.

Reading an identifier. The attacker runs an auction with their own domain participating as the only buyer. The browser triggers a `GET` request to the `trustedBiddingSignalsUrl` and includes the names of all interest groups in the parameters. The attacker's server can then reconstruct the identifier by mapping the names of observed interest groups to a value of 1 and filling all other bits with a value of 0. This attack functions in a similar manner to the one presented in Figure 5, but reading an identifier only requires a single auction, and the identifier is read by the attacker's BYOS server.

Optimization. This attack can be optimized by storing the entire 32-bit identifier in a single interest group, by appending it in the `trustedBiddingSignalsUrl`. Thereafter, the identifier can be read when the browser initiates a `GET` request to the endpoint specified in the `trustedBiddingSignalsUrl` field, and does not need to be reconstructed from the names of multiple interest groups. This optimization is possible because of the trivial cost on the attacker to create new, untrusted key/value services on existing servers in the BYOS model.

Mitigation. Adding support for the BYOS model in its public release opens the API to misuse with new tracking vectors. The current infrastructure enables untrusted servers to log event-level data, i.e., the services can store and share details of individual requests with their servers to enable user tracking. When a requirement for the TEE-based key-value services is enforced, browsers will only send bidding signals to trusted servers. These servers will be required to not log event-level data or share details about individual requests they receive from browsers to any external servers.

Non-aggregated Win Reporting. The Privacy Sandbox includes functionality implemented by the Private Aggregation API, which ensures that events from bids and ad impressions are aggregated and anonymized before they are shared with advertisers. However, since the Private Aggregation API and its corresponding infrastructure are under development, the public release of the FLEDGE API adopts granular reporting. The triggers a network request that reports the winner of each auction back to the advertiser and includes the `name` of the winning interest group in these requests. An attacker can exploit this mechanism to track users by curating multiple auctions, with each one revealing one bit of information.

Writing an Identifier. The attacker can create a 32-bit identifier by generating 32 unique strings and creating interest group objects that share the same `owner` domain for strings that map to a value of 1. Additionally, within the bidding logic (made available at the link under `biddingLogicUrl`), the attacker exposes two functions: `generateBid()` which includes logic to create a bid, and `reportWin()` which can receive details about the bid upon winning an auction. Note that the `reportWin()` function is invoked by additionally calling `sendReportTo()` within the `generateBid()` function.

Reading an Identifier. The attacker runs multiple single-buyer auctions. With each auction, the browser reports the winner by calling the interest group's `reportWin()` function, which in turn triggers the `sendReportTo()` function and provides it with the name of the winning interest group. The attacker can ensure that each auction is won by a different interest group by keeping track of previous winners and passing their names in seller-provided auction signals. Once all stored interest groups have won at least one auction, the attacker's bidding logic returns a null value. Thereafter, all interest groups that won an auction can be considered to correspond to a bit value of 1 and the user's identifier can be recreated. This attack is designed in a similar manner to the one presented in Figure 5. However, each bit is read based on FLEDGE's current reporting mechanism instead of its ad rendering setup.

Mitigation. The attack relies on a measure that currently bypasses aggregation of ad impressions and allows advertisers to receive granular information about individual auctions. This can be limited by introducing the Private Aggregation API which will anonymize auction events before reporting them to the advertiser, thereby limiting their ability to synchronously gather and construct identifiers across multiple auctions.

Non-aggregated Event-level Reporting. FLEDGE also allows advertisers to gather events within rendered ads, including ad impressions and clicks. Fenced frames include functionality to support event-level reporting through the new `window.fence.reportEvent()` API call that is only triggered within the frame's context and handled by the browser. While events reported from fenced frames are intended to be anonymized with the Private Aggregation API, FLEDGE currently allows advertisers to gather information from individual ads.

Writing an identifier. The attacker can embed a unique, 32-bit identifier in the interest group's `userBiddingSingals` field. This field is made available to the advertiser's bidding logic when the interest group participates in an auction.

Reading an identifier. In subsequent sessions, the attacker runs an auction with itself as the only participating advertiser. The browser executes the advertiser's bidding logic, making the identifier from the previously defined `user-`

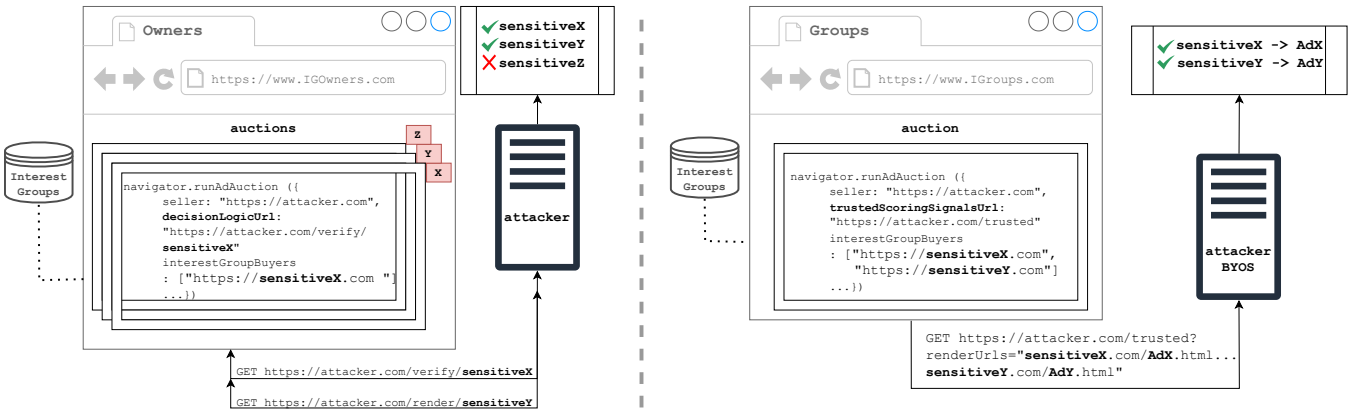

Figure 6: Cross-site leaks of owners (left) and interest group contents (right).

`BiddingSignals` available to the `generateBid()` function. The advertiser can place an infinitesimal positive bid value (FLEDGE limits bid precision values to an 8-bit mantissa and 8-bit exponent [44]) and specify an `adcost` of 0 to avoid costs. The identifier is returned to the seller as part of the `adMetadata`. Once the seller receives bids, the browser executes its `scoreAd()` function, within which it extracts the identifier from the `metadata` and designates it as a `desirability` score. The identifier embedded in the `desirability` score is available to the `reportResult()` function, from which the seller can pass it back to the server as a network request by using one of the following methods. First, it can create a custom event, say *track*, which is then reported as an event from within the fenced frame using `reportEvent()`, which generates a network request to the attacker's domain with the identifier embedded as a query. Alternatively, the attacker can use a temporary reporting mechanism, `sendReportTo()`, which passes similar reports to a URL specified by the seller without relying on the fenced frame's functionality. In addition to its design similarity to the win reporting attack described above, this attack additionally relies on code being shared between an advertiser and a seller, i.e., it assumes that the two parties either collude or a single entity participates as both. This assumption is reasonable, as we further discuss in §5.

Mitigation. This attack vector also relies on a measure that currently bypasses aggregation of ad impressions and allows advertisers to receive granular information about individual auctions, and can be prevented by ensuring that ad impressions are aggregated and anonymized with the Private Aggregation API before sending them out of the browser.

4.2 Cross-site Data Leaks

Gathering Interest Group Owners. First, we describe a technique that allows arbitrary sellers to learn the list of domains that have added that specific browser to an interest group. Our method exploits the default mechanisms with which the FLEDGE API handles auctions.

Existing exploitable data. When a user visits a website, say `example.com`, the website can add the user to an interest group that captures specific user interests corresponding to the user's actions on the site. Note that both top-level domains and third-party advertisers (embedded within frames or through script tags) can add a user to an interest group. In either case, the entity that created the interest group will be considered the owner.

Preparation. The attacker prepares a pre-defined list of domains that it intends to test against users. If this list includes "sensitive" domains, the presence of an interest group from specific domains can leak private or personal information about the user. In general, knowledge of whether the user has visited domains that are of interest to the attacker can be leveraged for a variety of privacy-invasive attacks depending on the nature of those domains (e.g., targeting users of competing businesses, targeting users with specific interests, extortion [46]).

Observing the presence of an interest group owner. When a user visits the attacker's website, it can execute auctions so as to infer the presence of an interest group owner. Specifically, for each domain in the pre-defined list of target domains, the website defines a new, single-buyer auction. The auction configuration includes a `decisionLogicUrl`, which is an attacker-controlled link that receives bid responses and decides the winner of the auction. However, if none of the buyers participating in an auction have an interest group stored in the browser, the network request to the `decisionLogicUrl` will not be triggered. This reveals the presence or absence of interest groups from each targeted owner to the attacker. Figure 6 (left) presents an overview of this attack.

Performance evaluation. The attack is scalable to a large list of pre-defined domains. For our evaluation we gathered a list of 500 domains and used Puppeteer to open a fresh browser instance and join interest groups from a subset of owners. Following the join operation, we navigated to a third-party attacker domain which executed the sniffing attack against the

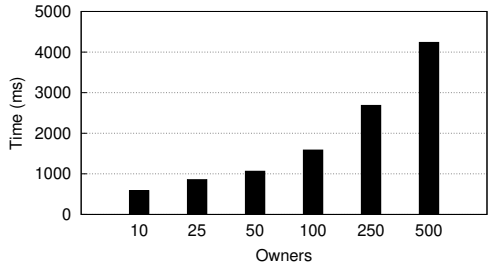

Figure 7: Performance of interest group owner-sniffing.

500 domains. For each subset size we repeated the experiment 10 times and gathered the average observed time. As can be seen in Figure 7, the attack duration depends on how many of the interest group owners are present in the user's browser. When all 500 owners were present, the attack took approximately 4.2 seconds to complete. It is important to note that the attack is transparent to the user since FLEDGE operations are performed within dedicated worklets outside of the main thread [44], and can be further scaled up based on the attacker's list of targeted owners. The attack was also fully accurate in all of the experimental runs (i.e., no false positives or negatives).

Multiplexing. We performed this attack with HTTP/2 connections. Since the browser only allows 6 HTTP/1.1 connections to be opened simultaneously, we multiplexed HTTP/2 connections to allow for a large number of concurrent streams over a single connection [47–49].

Mitigation. The attack is made possible by the synchronous approach and logic applied by the FLEDGE API before triggering requests to the `decisionLogicUrl`. This attack can be mitigated with a two-step approach; first, the browser would need to asynchronously gather and cache the decision logic code for a pre-approved list of sellers to prevent network requests from being triggered for each auction. Second, the decision logic needs to be restricted in its ability to trigger network requests outside of the current auction's context.

Interest Group Leaks. While the previous attack allows an attacker to learn the presence of interest group owners, it is unable to gather details about the contents of the interest groups stored by each owner. Here we present an additional attack that builds upon the previous attack for obtaining links to ads stored within individual interest groups within the `renderUrl` field. Gathering ads corresponding to the field helps an attacker learn details about user interests and browsing behavior towards which the specific ad is geared towards. This is done by running an additional auction with the list of previously identified owners as participants. When a browser observes this auction, it gathers bid responses from each of the interest group owners. Subsequently, along with the positive bids it will pass the `renderUrl` for each positive bid to the seller at the link defined in `trustedScoringSignalsUrl` for that auction. This functionality aims to assist the seller in deciding the winner of the auction.

Prerequisites. Within each interest group, the owner must have specified a valid `renderUrl`. When participating in an auction that is run by the attacker, we assume that the owner places a positive bid towards at least one interest group.

Trusted Scoring Signals. Similar to the previously described use of trusted bidding signals being passed to advertisers, the FLEDGE API includes an equivalent mechanism to help sellers gather information that can be factored in determining the outcome of an auction. These signals are intended to be made available through a secure key/value service and provided to the decision logic within a TEE-environment. However, in order to support BYOS models, the API provides the seller with the `renderUrl` along with individual bid responses for each advertiser that placed a positive bid within the auction, thereby enabling a cross-site leak of the contents of individual interest groups. Figure 6 (right) presents an overview of this attack.

Gathering Ad Links. Once the attacker has a list of domains for which it intends to gather interest group information, it creates an auction including the list of owners as participants. It ensures that the auction configuration defines a `trustedScoringSignalsUrl` that points to a service under the attacker's control. Upon running the auction, the browser collects the `renderUrl` from every interest group owner that placed a positive bid in the auction, and passes these links as parameters to a GET request sent to the attacker's trusted signal server. As such, the attacker can gather the `renderUrl` corresponding to one interest group per owner. Each `renderUrl` reveals the specific interest towards which the interest group is targeting the user. These links can leak specific information about a user including products that they have interacted with (e.g., specific hotel and flights that they are attempting to book) or even sensitive interests (e.g., products from sensitive sites that the advertiser is remarketing).

Mitigation. Adding support for the BYOS model in the public release of FLEDGE opens the API to misuse not only from advertisers but also from sellers attempting to infer details about a user's browsing activity. This can be mitigated through the use of an established TEE environment that would operate without the need for an individual `renderURL` to be passed to the seller, in an unsafe manner, for every bid.

4.3 Service Disruption

Browser Crash. Prior to its public release, we observed an out-of-bounds error that could be triggered in FLEDGE APIs, thereby causing the browser to abruptly terminate all on-going sessions. The vulnerability was the result of incomplete error handling in the event of a prioritized interest group exiting an auction before producing a valid bid.

Preparation. The attacker adds the browser to two interest groups with a specific composition of fields. The first group contains values for `name`, `biddingLogicUrl`, and `ads`. In addition to these fields, the second group also defines `trustedBiddingSignals` and is prioritized by setting the `enableBid-`

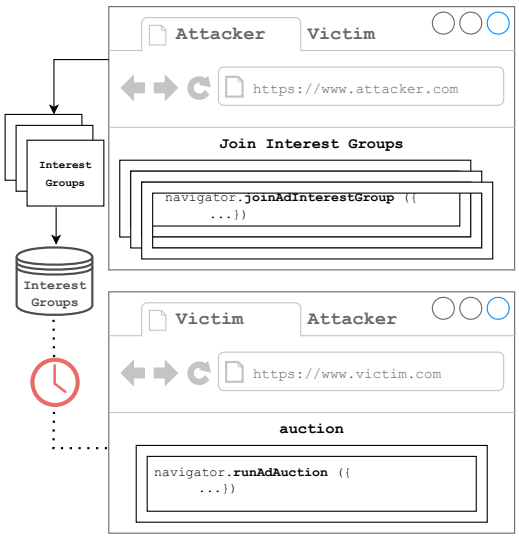

Figure 8: Blocking Ad Auctions.

`dingSignalsPrioritization` field to `True`. Finally, the `biddingLogicUrl` of the second group needs to return invalid bidding logic, like an empty or non-JavaScript response.

Triggering the crash. After making the browser join the two specially-crafted interest groups, the attacker can run an auction that only involves those two groups. The crash is triggered after an non-deterministic number of auction invocations and, therefore, is executed multiple times in a loop.

Bidding pipeline details. When running an auction, the FLEDGE API invokes and executes the bidding logic for both interest groups. However, note that the second invocation does not wait for the first invocation to complete, and only the order is determined by prioritization. Since the second interest group includes an invalid URL in the `biddingLogicUrl` field, the corresponding worklet crashes and is filtered out of the auction. However, upon doing so it fails to update an auction variable that tracks the expected number of bid responses. Thereafter, once the first interest group completes its bid, the browser attempts to read the result of an additional, non-existent bid, which results in an unhandled out-of-bounds exception that eventually crashes the browser.

Mitigation. we reported our findings to the Chromium team and they promptly fixed the bug prior to the public release of FLEDGE, by including a test case for the reported scenario to ensure that the error is promptly handled.

Blocking Ad Auctions. FLEDGE attempts to limit the impact that ad auction operations performed by one advertiser can have on operations performed by others. Figure 8 presents an overview of this attack.

Existing protections. The browser ensures that all FLEDGE operations performed by a domain are executed within a worklet dedicated to that domain. This restriction ensures that the operations of one domain do not impact the actions of other domains. Additionally, the browser ensures the timely completion of an auction by enforcing a 500ms timeout on the execution of the bidding logic for each buyer that is participating in an auction. However, regardless of the dedicated worklets, all FLEDGE-based operations interact with a global interest group database, access to which is not thread-safe, and on-going operations on the database can block access until the operations complete [50]. We note that while there is existing support for per-partition storage of interest groups [51], the official public release is vulnerable to the attack that we describe next.

Prerequisites. We assume that the user has visited two domains in different tabs within the same browsing window. The attacker is embedded as an advertiser on both domains. In the first tab, the attacker makes a large number of calls to `navigator.joinAdInterestGroup()`. In the second tab, an arbitrary seller runs an ad auction after the attacker has already issued a large number of calls to join new interest groups.

Blocking. When the attacker makes a large number of calls to `navigator.joinAdInterestGroup()`, each of these calls are queued to execute within the attacker's dedicated worklet. However, these operations also access and update the global interest group database associated with the browser's profile. When a seller starts an ad auction, the browser also queues a call for the bidding and decision logic to execute within other worklets, but their execution waits for access to the interest groups database, thus delaying the completion of their auction.

Attack details. In our experiments we found that by invoking 10,000 calls to `navigator.joinAdInterestGroup()` the attacker can prevent an ad from rendering for up to 25 seconds. The ad ecosystem benefits from displaying relevant ads to users in a timely manner that seamlessly integrates with their browsing session. Depending on the placement of the ad on the page, and the amount of time the user actually spends on that specific page, the attack may result in the user not even seeing any ads. Preventing the timely execution of ad auctions can prevent competitor advertisers, sellers, and publishers from gathering monetary gains from ad impressions or conversions. Performing this attack at a larger scale across multiple users and devices can significantly impact the revenue streams of participating entities.

Mitigation. The attack exploits the browser's lack of checks for locks placed on global resource pools. Implementing a per-origin limit on access to the interest group database and a use of caches for FLEDGE operations per-tab, can limit the delay introduced by one domain's operations on others.

4.4 Pollution Attacks

Polluting DoubleClick Interest Groups. Throughout 2023, Google Ad Manager introduced FLEDGE-based auctions to a small set of publishers, with the intention of scaling these auctions to 10% of all ads that they serve [37, 38]. As a result, during our measurement period, we observed the creation and use of a large number of interest groups owned by `doubleclick.net`.

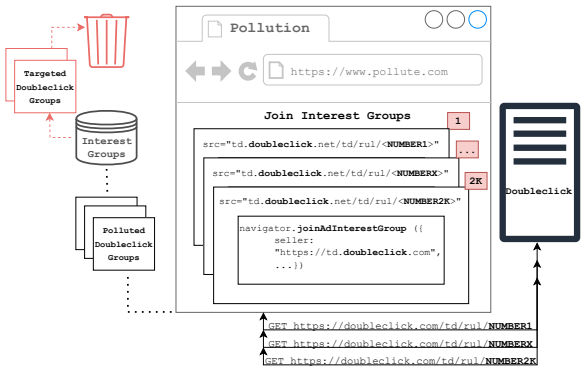

Figure 9: Polluting Doubleclick Interest Groups.

Specific URL Patterns. We noted that Google Ad Manager added interest groups from pages with embedded JavaScript code that, when loaded, automatically added the browser to interest groups. The frames were loaded from a URL with a common pattern, i.e., `td.doubleclick.net/td/rul/<NUMBER>`, where the `<NUMBER>` indicated a unique, 10-digit number corresponding to a specific interest group. Upon visiting these links, we observed that Google did not apply checks and authentication, and responded to all incoming requests without verifying their origin. The value of the `<NUMBER>` corresponds to the name of the Google-owned interest group that the browser will be added to. Using this approach, the names and purpose of the interest group is obfuscated, making it difficult to infer the specific user interest to which the group corresponds.

Per-origin Interest Group Limits. In order to limit the storage and compute resources taken by each advertiser, the Privacy Sandbox team initially placed a limit on the number of interest groups into which each interest group owner can place a browser to a total of 1K at the time of our experimentation (to accommodate negative interest groups, this was raised to 2K [43]). Once an interest group owner reaches this limit, joining a new interest group will result in the removal of the oldest interest group for that owner present within the browser.

Pollution. When a user visits a site containing code from the attacker's domain, it can create 1K frames (currently, 2K) that load from Google Ad Manager, each with the `src` attribute set to `td.doubleclick.net/td/rul/<NUMBER>`, where `<NUMBER>` corresponds to a different interest group. We compiled a list URLs captured during our measurement period. Making calls to each of these links results in the browser joining 1K new interest groups and removing pre-existing interest groups owned by Google Ad Manager. Figure 9 presents an overview of this attack.

Caveats. Some interest groups may be deprecated or invalidated over time; in practice, the attack may need to embed more than 1K frames.

Implication. While our attack prototype is specific to Google Ad Manager, it offers insight into potential pitfalls in the general adoption of FLEDGE. Specifically, the onus of enforcing access control to FLEDGE-related operations and preventing accidental, unauthenticated triggers of such code currently lies on the advertisers. In the absence of such protections, malicious attackers can deploy such attacks to influence bids placed by the advertiser, directing them to incorrectly target users and, as a result, observe a decline in their ad revenues. While in the case of Google Ad Manager the URL patterns corresponded to seemingly arbitrary or obfuscated interest groups, as the `<NUMBER>` does not directly reveal specific user interests, the attack can be fully targeted in cases where that information can be inferred – this could be achieved by employing a system like AdFisher [4] and mapping specific user interests to specific interest groups. The attacker can then pollute an advertiser's interest group set by adding the browser to interest groups corresponding to incorrect user interests or large campaigns, thereby having the advertiser serve irrelevant ads to users (including incorrect political ads and misinformation campaigns) at a large scale.

Mitigation. The backend server managed by Google Ad Manager (or any advertiser that stores interest groups in the user's browser) should implement authentication and verification mechanisms for validating incoming requests triggered from untrusted origins. Additionally, the browser can add checks to ensure that loaded HTML does not automatically execute code that adds users to a large number of interest groups.

5 Discussion and Limitations

Ethics and disclosure. Our analysis was conducted using test devices, accounts and websites. We did not include any users in our experiments. Moreover, due to the severe implications of the flaws uncovered by our research, we responsibly disclosed our findings to Google upon their discovery. In total, we have submitted 12 reports to Google, detailing our techniques and providing the necessary information to replicate and verify our attacks. At the time of this writing, Google has patched the issue that enabled the DoS attack that crashes the browser. They have also awarded us through their vulnerability report program for the pollution attack, which is being investigated. Additionally, they are currently investigating one cross-site leak and one tracking attack. We have not received feedback about the remaining attacks.

Attack practicality. The current public-release versions of Google Chrome are vulnerable to the attacks we present in §4 (apart from the "browser crash" attack that was patched after our disclosure). The user tracking attacks require the attacker to create single-buyer auctions and to participate as both an advertiser and a seller. Based on our observations (see §3), such dual participation in auctions is reasonable and found in the wild (e.g., Doubleclick). Importantly, FLEDGE does not restrict entities from participating in both roles, and also does not restrict single-buyer auctions. We tested cross-site leak attacks against public domains that we own, and in Figure 7, showed that 500 domains can be sniffed in 4.2 seconds. We also confirmed the service disruption attacks with the Chromium team.

Severity. The attacks we present in §4 assume a weak attacker, that can even be embedded as a third-party entity on a website that a user visits. In §3, we show that popular ad networks (e.g., Criteo, Teads, Amazon), i.e., third-party entities, use the FLEDGE API instead of the top-level domains themselves. As shown in Table 3, our attacks violate *all* the key privacy advancements detailed in the FLEDGE proposal. Additionally, our service disruption attacks present additional risks to both users and the ecosystem at large.

Evolution and planned mitigations. FLEDGE was officially released to the public in Chrome v117 in September 2023, even though its current implementation does not include all of the components detailed in Google's original designs. As a result, certain features or capabilities previously reported are not yet available in the stable release. Surprisingly, Google opted to officially deploy FLEDGE *without* having implemented components that are integral to some of the privacy protections they have announced (e.g., k-anonymity which was intended for late Q3 2023 but has yet to be released [52]). While some of our novel attacks leverage components or functionalities that are "temporary" until replacement components are deployed, other attacks will remain unaffected (see Table 3).

Crucially, while most of our attacks were disclosed *prior* to the public release of FLEDGE, Google opted to continue with the public release without any appropriate mitigations being put in place (apart from patching the issue that enabled our browser crash). As a result, users are currently facing significant additional privacy risks compared to the pre-FLEDGE ad auction ecosystem (e.g., arbitrary websites being able to infer part of the user's browsing history, or pollute their ad recommendations). We argue that mechanisms that aim to completely reshape an entire ecosystem should not be publicly deployed prior to implementing all announced safeguards or addressing major flaws that have been disclosed to the vendor.

K-anonymity. According to documentation, FLEDGE will employ k-anonymity to strengthen. For instance, *a "crowd of 50 users per creative over 7 days*" will be required for rendering the creative (i.e., ad). Moreover, requests will be routed through Oblivious HTTP, which prevents the server from linking requests to a specific client or identifying requests as having come from the same client [33]. Since attackers can trivially bypass that restriction by deploying numerous browser instances, and currently no robust mechanism exists for verifying that requests are from legitimate Chrome instances [53], users will be required to be logged into their Google accounts. In practice, however, this does not pose a major obstacle to attackers; prior work has demonstrated systems for automating the account creation process [54, 55], and solving CAPTCHAs [56, 57], and has studied underground markets that emerge for assisting attackers in passing challenging account verification constraints (e.g., phone verification [58]). As such, even if FLEDGE employs k-anonymity, it will not be sufficient for fully protecting users' privacy. Importantly, we also demonstrate tracking techniques that leverage fields that *will not* be protected by k-anonymity.

6 Related Work

To the best of our knowledge, our work presents the first in-depth investigation of FLEDGE, resulting in a series of novel attacks with severe privacy implications. Next, we provide an overview of prior research on web tracking and privacy-invasive attacks, as well as explorations of the ad ecosystem.

Privacy Sandbox. Browser vendors and consumer protection organizations have raised concerns about the privacy implications of Google's Privacy Sandbox. Recently, Ali et al. [59] presented an automated framework, CanITrack, for testing browser mechanisms as potential tracking vectors, and demonstrated various tracking techniques including two that misuse Privacy Sandbox APIs (i.e., Trust Token API, FLEDGE API). That study presents a tracking attack against FLEDGE that is conceptually similar to some of the attacks we present and to prior work in user tracking [60,61]. Their attack relied on collusion between multiple advertiser domains and only analyzed a single aspect of FLEDGE auctions, i.e., responses to unsuccessful auctions. Instead, our techniques leverage different FLEDGE functionalities. However, CanITrack cannot identify the 5 non-tracking attacks nor automatically design or find the 7 tracking attacks we present in §4. The framework requires the *manual* development of the client-side and server-side script snippets to identify the fields that we intend to target (e.g., biddingLogicURL, biddingWasmHelperUrl, etc.). Nonetheless, once those have been created it can identify the effectiveness and extent of the tracking potential (e.g., across third-party contexts, across regular and private browsing modes). Based on our analysis which covers these dimensions, all of our tracking attacks function cross-site, using embedded third-parties. Additionally, since FLEDGE APIs cannot be called in the private browsing mode, our attacks only work during regular browsing sessions. Finally, clearing all browsing data will also result in interest groups being cleared from the browser.

Furthermore, we conduct an extensive, in-depth analysis that uncovers a series of severe design and implementation flaws that enable several different classes of attacks. Previously, Berke and Calacci [62] studied Google's FLoC (which has been cancelled and replaced by "Topics"), a system intended to group users with common browsing behaviors, and found that the succession of FLoC's k-anonymous cohort IDs over time could lead to tracking. Beugin and McDaniel [63] analysed the Topics API using statistical models of the distribution of user behaviors, and reported weaknesses in the proposal's privacy guarantees that can leak user interests and also help trackers re-identify individual users.

Tracking. Numerous prior studies have explored and measured mechanisms and web practices that relate to tracking. Lerner et al. [64] used the Internet Archive to provide a retrospective analysis of third-party tracking over time,

while Englehardt and Narayanan [15] presented a large-scale comprehensive measurement of online tracking practices. Franken et al. [65] uncovered flaws in browsers' and browser extensions' implementations of various third-party cookie policies that enabled tracking and other cross-site attacks, while Dimova et al. [16] studied how CNAME DNS records were being used to bypass anti-tracking defenses. More recently, studies have demonstrated cookieless techniques for storing tracking identifiers [59], such as using HSTS policies [60] and favicons [60]. In a different line of work, numerous studies have explored various aspects of browser fingerprinting [66–76] as it can allow websites to track users based on unique characteristics of their devices.

Ad ecosystem. Tracking and advertising are inextricably connected and prior work has extensively explored and measured the complexities of the advertising ecosystem [3, 77–79]. Digital ad spaces are bought and sold through a real-time bidding process that incentivizes advertisers to gather granular user information to help them place competitive bids. Zeng et al. [80] gathered evidence of the use of individual user profiles in real-time bidding strategies. Papadopoulos et al. [81] presented a method to evaluate user profiles and their influence on advertising bids. Pachilakis et al. [82] studied the adoption of header bidding on the web and reported on the dominance of a few advertisers. In another study, Pachilakis et al. [83] observed that the availability of user demographics increased ad prices, explaining the incentives behind user tracking. Alternatively, Venkatadri et al. [84] demonstrated how Facebook's "custom audience" functionality could be misused for inferring users' PII and activities, and Vines et al. [9] showed how ad targeting could be used for tracking users' location. Accordingly, a large body of work has proposed techniques for detecting and blocking ads and trackers [85–89], while prior work has also proposed various schemes for privacy-preserving ads [25, 90–93].

7 Conclusion

Google recently announced plans to phase out support for third-party cookies in Chrome, with the goal of mitigating cross-site tracking and strengthening user privacy. However, Google's revenue stream relies *heavily* on the online ad ecosystem, thus introducing an obvious dilemma. Accordingly, Google has started rolling our the Privacy Sandbox, a collection of new browser mechanisms and standards that aim to provide privacy-preserving alternatives to existing processes and workflows in the online ad ecosystem. Due to Google's pivotal role and positioning in the web ecosystem (i.e., running the most prevalent browser, search engine, and ad display network), these plans have massive implications for online privacy. This has led to serious concerns being raised by other browser vendors and user protection organizations. Putting concerns about Google's monopoly aside, our in-depth empirical analysis of the FLEDGE API demonstrates that the Privacy Sandbox initiative is currently exposing users to significant privacy risks. We have responsibly disclosed our findings in an effort to assist Chrome in improving their system and better protecting users, and hope that our research will enable and motivate additional research into Chrome's emerging APIs.

Acknowledgements

We thank the anonymous reviewers and shepherd for their helpful feedback. This project was supported by the National Science Foundation (CNS-2211574, CNS-2143363). The views in this paper are only those of the authors and may not reflect those of the US Government or the NSF.

References

[1] "Bcc research expects the digital advertising space to surpass the trillion-dollar mark by 2027," 2023. [Online]. Available: https://blog.bccresearch.com/digital-advertising-industry

[2] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten, "Cookies that give you away: The surveillance implications of web tracking," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 289–299.

[3] M. B. Musa and R. Nithyanand, "Atom: ad-network tomography," *Proceedings on Privacy Enhancing Technologies*, vol. 4, pp. 295–313, 2022.

[4] A. Datta, M. C. Tschantz, and A. Datta, "Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination," *arXiv preprint arXiv:1408.6491*, 2014.

[5] M. Lécuyer, G. Ducoffe, F. Lan, A. Papancea, T. Petsios, R. Spahn, A. Chaintreau, and R. Geambasu, "Xray: Enhancing the web's transparency with differential correlation," in *USENIX Security*, 2014.

[6] A. Korolova, "Privacy violations using microtargeted ads: A case study," in *IEEE ICDM Workshops*, 2010.

[7] B. Imana, A. Korolova, and J. Heidemann, "Auditing for discrimination in algorithms delivering job ads," in *Web Conference (WWW)*, 2021.

[8] C. Castelluccia, M.-A. Kaafar, and M.-D. Tran, "Betrayed by your ads! reconstructing user profiles from targeted ads," in *PETS 2012*.

[9] P. Vines, F. Roesner, and T. Kohno, "Exploring adint: using ad targeting for surveillance on a budget-or-how alice can buy ads to track bob," in *WPES*, 2017.

[10] M. Ali, A. Goetzen, P. Sapiezynski, E. Redmiles, and A. Mislove, "All Things Unequal: Measuring Disparity of Potentially Harmful Ads on Facebook," in *ConPro*, 2022.

[11] M. Ali, P. Sapiezynski, M. Bogen, A. Korolova, A. Mislove, and A. Rieke, "Discrimination through optimization: How facebook's ad delivery can lead to biased outcomes," in *CSCW*, 2019.

[12] O. Papakyriakopoulos, C. Tessono, A. Narayanan, and M. Kshirsagar, "How algorithms shape the distribution of political advertising: Case studies of facebook, google, and tiktok," in *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, 2022, pp. 532–546.

[13] "Webkit - intelligent tracking prevention 2.2," 2019. [Online]. Available: https://webkit.org/blog/8828/intelligent-tracking-prevention-2-2/

[14] "Brave - advanced privacy," 2023. [Online]. Available: https://brave.com/privacy-features/

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

[16] Y. Dimova, G. Acar, L. Olejnik, W. Joosen, and T. Van Goethem, "The cname of the game: Large-scale analysis of dns-based tracking evasion," *arXiv preprint arXiv:2102.09301*, 2021.

[17] G. Franken, T. Van Goethem, and W. Joosen, "Who left open the cookie jar? a comprehensive evaluation of third-party cookie policies," in *USENIX Security Symposium*, 2018.

[18] I. Sanchez-Rola, X. Ugarte-Pedrero, I. Santos, and P. G. Bringas, "The web is watching you: A comprehensive review of web-tracking techniques and countermeasures," *Logic Journal of the IGPL*, vol. 25, no. 1, pp. 18–29, 2017.

[19] "Mozilla - competition should not be weaponized to hobble privacy protections on the open web," 2022. [Online]. Available: https://blog.mozilla.org/netpolicy/2022/04/12/competition-should-not-be-weaponized-to-hobble-privacy-protections-on-the-open-web/

[20] "Brave - privacy and competition concerns with google's privacy sandbox," 2022. [Online]. Available: https://brave.com/web-standards-at-brave/6-privacy-sandbox-concerns/

[21] "Eff - don't play in google's privacy sandbox," 2019. [Online]. Available: https://www.eff.org/deeplinks/2019/08/dont-play-googles-privacy-sandbox-1

[22] "What is the privacy sandbox?" 2021. [Online]. Available: https://developer.chrome.com/docs/privacy-sandbox/overview/

[23] "Protected audience api," 2022. [Online]. Available: https://developer.chrome.com/docs/privacy-sandbox/protected-audience/

[24] "FLEDGE Code Repository," 2024. [Online]. Available: https://github.com/masood/fledge-sec-24

[25] K. Zhong, Y. Ma, and S. Angel, "Ibex: Privacy-preserving ad conversion tracking and bidding," in *ACM CCS*, 2022.

[26] M. Squarcina, P. Adão, L. Veronese, and M. Maffei, "Cookie crumbles: breaking and fixing web session integrity," in *USENIX Security Symposium*, 2023.

[27] "Private aggregation api," 2022. [Online]. Available: https://developers.google.com/privacy-sandbox/relevance/private-aggregation

[28] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security vulnerabilities of sgx and countermeasures: A survey," *ACM Comput. Surv.*, vol. 54, no. 6, jul 2021. [Online]. Available: https://doi.org/10.1145/3456631

[29] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, "Hacking in darkness: Return-oriented programming against secure enclaves," in *26th USENIX Security Symposium*, 2017.

[30] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium*, 2018.

[31] J. Cui, J. Z. Yu, S. Shinde, P. Saxena, and Z. Cai, "Smashex: Smashing sgx enclaves using exceptions." ACM CCS, 2021.

[32] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "VoltPillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface," in *30th USENIX Security Symposium*, 2021.

[33] M. Thomson and C. Wood, "Rfc 9458 oblivious http," 2023.

[34] "Original-TURTLEDOVE," 2021. [Online]. Available: https://github.com/WICG/turtledove/blob/main/Original-TURTLEDOVE.md

[35] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," ser. NDSS 2019.

[36] G. A. Johnson, S. K. Shriver, and S. G. Goldberg, "Privacy and market concentration: intended and unintended consequences of the gdpr," *Management Science*, 2023.

[37] "Protected Audience API and Ad Manager after Chrome GA - Google Ad Manager Help." [Online]. Available: https://support.google.com/admanager/answer/13627134?hl=en

[38] "Ad Manager testing updates - Google Ad Manager Help." [Online]. Available: https://support.google.com/admanager/answer/13178817?hl=en#zippy=%2Cpublished-on-january-initial-privacy-sandbox-tests-are-encouraging-with-more-insights-to-come-this-year

[39] "Fledge tester list." [Online]. Available: https://github.com/WICG/turtledove/blob/main/fledge-tester-list.md

[40] "Google's post-cookie ad tech tests face skepticism." [Online]. Available: https://adage.com/article/digital-marketing-ad-tech-news/googles-post-cookie-ad-tech-tests-face-skepticism/2430111

[41] M. Ruminski, "Rtb house update on fledge tests." [Online]. Available: https://blog.rtbhouse.com/rtb-house-update-on-fledge-tests/

[42] Teads, "FLEDGE Key/Value Service." [Online]. Available: https://github.com/teads/fledge-key-value-service

[43] "Increase on device Interest Group cap to 2000 per owner · Issue #798 · WICG/turtledove." [Online]. Available: https://github.com/WICG/turtledove/issues/798

[44] "Protected Audience API (a.k.a. FLEDGE)." [Online]. Available: https://github.com/WICG/turtledove/blob/main/FLEDGE.md

[45] Google Campaign Manager 360 Help, "Helping advertisers comply with the GDPR & AADC." [Online]. Available: https://support.google.com/campaignmanager/answer/9028179?hl=en

[46] E. Maris, T. Libert, and J. R. Henrichsen, "Tracking sex: The implications of widespread sexual data leakage and tracking on porn websites," *new media & society*, vol. 22, no. 11, 2020.

[47] "Browser connection limitations | Documentation." [Online]. Available: https://docs.diffusiondata.com/cloud/latest/manual/html/designguide/solution/support/connection_limitations.html

[48] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," Internet Engineering Task Force, Request for Comments RFC 2616, Jun. 1999. [Online]. Available: https://datatracker.ietf.org/doc/rfc2616/

[49] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Engineering Task Force, Request for Comments RFC 7540, May 2015. [Online]. Available: https://datatracker.ietf.org/doc/rfc7540/

[50] Chromium, "Chromium Source: Interest Group Storage." [Online]. Available: https://source.chromium.org/chromium/chromium/src/+/main:content/browser/interest_group/interest_group_storage.h;l=33?q=THREAd&ss=chromium%2Fchromium%2Fsrc:content%2Fbrowser%2Finterest_group%2F

[51] ——, "Chromium Source: Interest Group Manager." [Online]. Available: https://source.chromium.org/chromium/chromium/src/+/main:content/browser/interest_group/interest_group_manager_impl.h

[52] "Chrome - protected audience api feature status," 2023. [Online]. Available: https://developer.chrome.com/docs/privacy-sandbox/protected-audience-api/feature-status/

[53] "Chrome - fledge k-anonymity server private information retrieval," 2023. [Online]. Available: https://github.com/WICG/turtledove/blob/main/FLEDGE_k_anonymity_server.md#private-information-retrieval

[54] K. Drakonakis, S. Ioannidis, and J. Polakis, "The cookie hunter: Automated black-box auditing for web authentication and authorization flaws," in *ACM CCS*, 2020.

[55] J. DeBlasio, S. Savage, G. M. Voelker, and A. C. Snoeren, "Tripwire: Inferring internet site compromise," in *IMC*, 2017.

[56] S. Sivakorn, I. Polakis, and A. D. Keromytis, "I am robot:(deep) learning to break semantic image captchas," in *2016 IEEE EuroS&P*, 2016.

[57] M. I. Hossen, Y. Tu, M. F. Rabby, M. N. Islam, H. Cao, and X. Hei, "An object detection based solver for Google's image reCAPTCHA v2," in *RAID 2020*.

[58] K. Thomas, D. Iatskiv, E. Bursztein, T. Pietraszek, C. Grier, and D. McCoy, "Dialing back abuse on phone verified accounts," in *CCS 2014*.

[59] M. M. Ali, B. Chitale, M. Ghasemisharif, C. Kanich, N. Nikiforakis, and J. Polakis, "Navigating Murky Waters: Automated Browser Feature Testing for Uncovering Tracking Vectors," in *NDSS 2023*.

[60] P. Syverson and M. Traudt, "Hsts supports targeted surveillance," in *USENIX FOCI*, 2018.

[61] K. Solomos, J. Kristoff, C. Kanich, and J. Polakis, "Tales of favicons and caches: Persistent tracking in modern browsers," in *Network and Distributed System Security Symposium*, 2021.

[62] A. Berke and D. Calacci, "Privacy limitations of interest-based advertising on the web: A post-mortem empirical analysis of google's floc," in *ACM CCS*, 2022.

[63] Y. Beugin and P. McDaniel, "Interest-disclosing mechanisms for advertising are privacy-exposing (not preserving)," in *PETS*, 2024.

[64] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016," in *USENIX Security Symposium*, 2016.

[65] G. Franken, T. V. Goethem, and W. Joosen, "Who left open the cookie jar? a comprehensive evaluation of Third-Party cookie policies," in *USENIX Security Symposium*, 2018.

[66] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *2013 IEEE Symposium on Security and Privacy*, 2013.

[67] P. Eckersley, "How unique is your web browser?" in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, ser. PETS'10, 2010.

[68] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in *Proceedings of W2SP 2012*, May 2012.

[69] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *Symposium on Security and Privacy (SP)*, 2016.

[70] O. Starov and N. Nikiforakis, "Xhound: Quantifying the fingerprintability of browser extensions," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 941–956.

[71] A. Sjösten, S. Van Acker, and A. Sabelfeld, "Discovering browser extensions via web accessible resources," in *CODASPY*, 2017.

[72] X. Lin, F. Araujo, T. Taylor, J. Jang, and J. Polakis, "Fashion faux pas: Implicit stylistic fingerprints for bypassing browsers' anti-fingerprinting defenses," in *IEEE Symposium on Security and Privacy*, 2023.

[73] K. Solomos, P. Ilia, N. Nikiforakis, and J. Polakis, "Escaping the confines of time: Continuous browser extension fingerprinting through ephemeral modifications," in *ACM CCS*, 2022.

[74] K. Solomos, P. Ilia, S. Karami, N. Nikiforakis, and J. Polakis, "The dangers of human touch: Fingerprinting browser extensions through user actions," in *USENIX Security*, 2022.

[75] S. Karami, P. Ilia, K. Solomos, and J. Polakis, "Carnus: Exploring the privacy threats of browser extension fingerprinting." in *NDSS*, 2020.

[76] S. Karami, F. Kalantari, M. Zaeifi, X. J. Maso, E. Trickel, P. Ilia, Y. Shoshitaishvili, A. Doupé, and J. Polakis, "Unleash the simulacrum: Shifting browser realities for robust {Extension-Fingerprinting} prevention," in *USENIX Security*, 2022.

[77] A. Andreou, M. Silva, F. Benevenuto, O. Goga, P. Loiseau, and A. Mislove, "Measuring the Facebook Advertising Ecosystem," in *Network and Distributed System Security Symposium*, 2019.

[78] J. Cook, R. Nithyanand, and Z. Shafiq, "Inferring tracker-advertiser relationships in the online advertising ecosystem using header bidding," *arXiv preprint arXiv:1907.07275*, 2019.

[79] M. A. Bashir, S. Arshad, W. Robertson, and C. Wilson, "Tracing information flows between ad exchanges using retargeted ads," in *USENIX Security Symposium*, 2016.

[80] E. Zeng, R. McAmis, T. Kohno, and F. Roesner, "What factors affect targeting and bids in online advertising? a field measurement study," in *ACM Internet Measurement Conference*, 2022.

[81] P. Papadopoulos, N. Kourtellis, P. R. Rodriguez, and N. Laoutaris, "If you are not paying for it, you are the product: How much do advertisers pay to reach you?" in *IMC*, 2017.

[82] M. Pachilakis, P. Papadopoulos, E. P. Markatos, and N. Kourtellis, "No more chasing waterfalls: A measurement study of the header bidding ad-ecosystem," in *IMC*, 2019.

[83] M. Pachilakis, P. Papadopoulos, N. Laoutaris, E. P. Markatos, and N. Kourtellis, "Youradvalue: Measuring advertising price dynamics without bankrupting user privacy," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 3, pp. 1–26, 2021.

[84] G. Venkatadri, Y. Liu, A. Andreou, O. Goga, P. Loiseau, A. Mislove, and K. P. Gummadi, "Privacy Risks with Facebook's PII-based Targeting: Auditing a Data Broker's Advertising Interface," in *IEEE Symposium on Security and Privacy*, 2018.

[85] U. Iqbal, C. Wolfe, C. Nguyen, S. Englehardt, and Z. Shafiq, "Khaleesi: Breaker of advertising and tracking request chains," in *USENIX Security Symposium*, 2022.

[86] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "Adgraph: A graph-based approach to ad and tracker blocking," in *IEEE Symposium on Security and Privacy*, 2020.

[87] S. Munir, S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso, "Cookiegraph: Understanding and detecting first-party tracking cookies," in *ACM CCS*, 2023.

[88] M. Ghasemisharif and J. Polakis, "Read between the lines: Detecting tracking javascript with bytecode classification," in *ACM CCS*, 2023.

[89] S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso, "Webgraph: Capturing advertising and tracking information flows for robust blocking," in *USENIX Security Symposium*, 2022.

[90] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy preserving targeted advertising," in *NDSS*, 2010.

[91] S. T. Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Privado: Privacy-preserving group-based advertising using multiple independent social network providers," *ACM TOPS*, vol. 23, no. 3, pp. 1–36, 2020.

[92] S. Guha, B. Cheng, and P. Francis, "Privad: Practical privacy in online advertising," in *USENIX NSDI*, 2011.

[93] M. Backes, A. Kate, M. Maffei, and K. Pecina, "Obliviad: Provably secure and practical online behavioral advertising," in *IEEE Symposium on Security and Privacy*, 2012.