Michael Kleber

Protected Audience API

★ W3C TPAC 2023, Seville, 11–15 September ★

The Problem We Are Trying To Solve

Context: Privacy Sandbox

Need to remove 3rd-party cookies, but that causes major publisher revenue loss

Goal

A way for ad tech to pick which ads to put on web pages, without needing or offering a way to recognize the same user across websites

Two Plausible Approaches

Approach #1: "The Easy Way"

Give the people who pick ads a little ad targeting info

Q: Can we fix this by the browser offering a little bit of information?

- Not enough to enable tracking
- Doesn't make users unhappy
- Lets websites recover a reasonable amount of revenue

A: We think so: Topics API

- Structurally familiar: Call an API, get some info, use the info
- Problem (with the whole approach): That info adds up over time

Approach #2: The Hard Way

Let the people who pick ads *use* some information *without revealing it*

Q: Is that even possible? A: We think so

Essential Ingredients:

- Some repository of "secret information"
- Some way to use the SI to choose an ad, without revealing SI
- Some way to render the chosen ad WORSI
- Some way for money to move around WORSI
- Some way for parties involved to learn what happened WORSI
- Some feedback mechanism for the ad choice (ML model) WORSI

The Privacy Sandbox Answers

We have two families of answers for how to do all of these

- The collection of proposals we are shipping now
  This has leaks —
  does not satisfy "WORSI" with an adversary trying to exfiltrate SI

- The future state that will have the desired privacy properties
  Still a lot of work to get there

The Protected Audience API

Our way to pick ads based on secret information

In incubation in WICG since 2020. What I will talk about today.

Other parts of Privacy Sandbox address others of those needs

- Rendering: Fenced Frames
- Learn What Happened: ARA (for attribution), PAA (for everything else)

Too hard to adopt everything all at once → incremental requirements

- Yes, that means things still leak, for now
- In the meantime, using these APIs is conditioned on a ***Public Attestation***

Protected Audience: The Interest Group

1. **A collection of (URLs of) ads the IG might want to show**
2. **(URL of) JS function that can produce a "bid" for an auction**
3. **Some locally-stored data — inputs to the JS function**
4. **Some keys to look up remotely-stored real-time data for the JS fn**
5. **A URL to enable periodic updates of 1–4 from a server.**

The Interest Group is just one piece of the whole Protected Audience API

- IG is often a wrapper around the ad tech that represents an advertiser
- Also much complexity in the choice of what ad will be displayed — wrapper around the ad tech that represents a publisher

1. URLs of ads the IG might want to show

Original design: IG contained Web Bundles, rendered with no network

Great data minimization, but navigable web bundles do not have much support

New plan: Good Old URLs (and mitigate timing attacks)

- Rendering in Fenced Frames, remove contact with the surrounding page
- k-anonymity of Render URL, weaken linkage between ad and viewer

We are in the market for a better solution!

2. JS function that produces a bid

Risk: Bidding JS sees both IG data and surrounding web page data

Original plan: On-device computation in an isolated worklet

- Problem 1: On-device? Some ad tech will want to use lots of compute
- Problem 2: Isolated worklet? Side channels

Early testing of a second option:
"Bidding & Auction Service", pushing JS work into a TEE

- Chrome-written open-source execution environment wrapping V8
- Hosted on a trustworthy cloud provider, TEE with remote attestation
- Only helps if you believe in TEEs on trustworthy cloud providers

3. Bids using some locally-stored data

Protected Audience is a purpose-built API that understands its data

This means the browser is in a position to be opinionated.

- Opinion: IGs contain data from only a single site
- Opinion: IGs can last for 30 days
- UX Implication: "Why did I see this ad?"
- UX Implication: "I don't like this ad"

4. Keys to look up remote real-time data

Ads bidding requires real-time data, e.g. "Do I have any money left?"

But the set of real-time data a browser is interested in is a fingerprint.

- Private Information Retrieval
- Real-time database is too large and changes too rapidly

Key-Value Lookup server, running inside a TEE

5. Updates of 1–4 from a server

"That thing you were looking at last week is now on sale!"

Original proposal: IG contains a URL, browser checks it once a day for updates

- Problem: IP address
- Problem: Time
- Mitigation: Instead of updating "in the background", do it while the user is on a page where that IG was invited to bid in an auction
- Mitigation: IP-blinding proxy? (cf. work in Anti-Fraud CG)
- Mitigation: Yet another TEE?

Protected Audience: The Interest Group

1. A collection of (URLs of) ads the IG might want to show
2. (URL of) JS function that can produce a "bid" for an auction
3. Some locally-stored data — inputs to the JS function
4. Some keys to look up remotely-stored real-time data for the JS fn
5. A URL to enable periodic updates of 1–4 from a server.