Manifest Unique ID - privacy & security considerations (PUBLIC)

phillis@google.com, dmurph@google.com
Last Updated: 08/25/2021

Reviewer	**Status:** 🕑Requested, 🙏 In Review, ✓ Approved
andzaytsev@google.com	✓ Approved

The manifest id member is supposed to be used to uniquely identify a web app, thus it should be set as a stable identifier and not carrying any user specific information. But since installed web apps are persisted per user profile, it can potentially be used to track users across browser sessions by detecting any behavior differences when the app is installed.
For example, **a malicious app can use the app id to carry user id, and try to detect if a user has a web app installed with the user id of a previous session**. This was raised during spec review. This doc analyzed possible scenarios and feasibility of such attacks and explained current mitigations.

Note: before this feature, start_url is used as the app identifier and the **same risks** are applied to the existing implementation.

Scenario with manifest update:

Manifest update depends on the id of the manifest matching the id of the installed app. If they do, then the update process occurs. If the update process is detectable by the site, this can theoretically be exploited to allow the attacking website to 'find' the id of a site after the user has cleared their browsing data.

Currently an app can detect a manifest update by seeing if app icons are fetched. App icons are fetched only after checking the id matches and manifest needs to be updated:

Example attack steps

1. User visits appA.
2. appA assigns user_id_1 to this user, and also dynamically sets id to user_id_1 in the manifest,
3. User installs appA.
4. User clears the site data and storage, but still remains in the current user profile.

5. User visits appA again.
6. appA guesses to set id to user_id_x in manifest as well as randomly changing another field to trigger a manifest update and see if a manifest update happens. If the guess is right, a manifest update will happen and the app can detect network requests made to app icons, thus the app successfully tracks the user between sessions.
7. If appA fails to guess the user, it sets a new user_id_2 and considers it a new user.

Attack feasibility & mitigation

An app's manifest is re-fetched on every page load. The number of guesses is limited by manifest update throttling, which we currently set to only allow manifest to be updated once per day. So at maximum, the app can only attempt to guess once. Also manifest update will only be performed if one of the fields in the manifest(excluding id) is changed, so the app will also need to change another field to a random value that has a low possibility of matching the field of the installed app.

Also worth noting that the similar risk is applied to other manifest fields(eg: start_url, scope, protocol_handlers, etc) that are used to detect manifests being changed and needing update. The logic is opposite here: if the value is changed(mismatch with installed app), an update will perform. So the app can set scope to user_id_x and detect if a manifest update doesn't occur, it could be a match. However because of the manifest throttling, this is much harder to be performed, because a non-update at most times only means it's throttled.

More Mitigation Ideas

Another further mitigation could be to always fetch all the icon data when fetching the manifest during page load to determine whether it needs to be updated. In this case, the app wouldn't be able to detect if a manifest update is happening.

Scenario with scope extensions:

https://github.com/WICG/manifest-incubations/blob/gh-pages/scope_extensions-explainer.md
Scope Extensions is currently under development. It enables a web application to capture user navigations to sites they are affiliated with. To enable the affiliation, the associated web app needs to specify a web-app-origin-association.json file to reference the other app. It is currently planned to use the app id to identify the app in the association file.
This can be exploited by guessing the app id in the association file, if the app id matches, the associated app can detect that the user can make successful navigation to the other app.

Example attack steps

1. User visits appA.

2. appA assigns user_id_1 to this user, and also dynamically sets id to user_id_1 in the manifest, as well as referencing it as user_id_1 in appB's web-app-origin-association.json.
3. User installs appA.
4. User clears the site data and storage, but still remains in the current user profile.
5. User visits appB.
6. appB guesses id to user_id_x and sets it in web-app-origin-association.json
7. appB links to appA in its content.
8. User clicks the link to launch appA if the guess is right.

Attack feasibility & mitigation

To have a successful attack, appB needs to have the user agent to perform an association file update and after the update is successful, the user needs to click the link to navigate to appA. The user will need to keep clicking links to re-attempt guesses, and for each re-attempt, the app needs to tell the user agent to re-fetch and update the association file.
This is mitigated the same way as the manifest update. The web app association information is updated during the manifest update(in OsIntegrationManager::UpdateUrlHandlers), so the same throttling is applied.