Web App Origin Association Design

This Document is Public

Authors: mandy.chen@microsoft.com, lu.huang@microsoft.com, akasamsetty@microsoft.com

Contributors/Reviewers: dominickn@chromium.org

Last Updated: November 2022

Table of Contents

Main Design Document

Overview

Following the suggestion from Add parser for web app origin association files (Id02648f1) · Gerrit Code Review (googlesource.com), this design document explains briefly the plan of implementing web app origin association parsing in a utility process and other association related work.

Utility process is a sandboxed, low-privilege, short-lived process that is used to do a specific task. Existing services that leverage utility processes can be found in chrome/utility/services.h.

Note: This document was originally written for web app origin association parsing being used to validate manifest URL handlers and has been adapted for scope extensions. This is a snapshot of the previous version of this document.

Folder Structure

Add a new folder components/webapps/web_app_origin_association for new files. This is where the main implementation lives.

```
components/webapps/services/web_app_origin_association
|--public
|    |--mojom
|         |--BUILD.gn
|         |--web_app_origin_association_parser.mojom
|
|--BUILD.gn
|--DEPS
|--web_app_origin_association_parser_impl.h
|--web_app_origin_association_parser_impl.cc
|--web_app_origin_association_fetcher.h
|--web_app_origin_association_fetcher.cc
|--web_app_origin_association_parser_service.h
|--web_app_origin_association_parser_service.cc

chrome/browser/web_applications/components
|--web_app_origin_association_manager.h
|--web_app_origin_association_manager.cc
```

Mojom Definitions

Set up utility process to parse web app origin association files (I04cec826) · Gerrit Code Review (googlesource.com)

components/webapps/services/web_app_origin_association/public/mojom/web_app_origin_association_parser.mojom

```
module web_app.mojom;

// The web app origin association structure is an internal representation of the
// web app origin association file described in the "Scope Extensions for Web
// Apps" explainer:
https://github.com/WICG/manifest-incubations/blob/gh-pages/scope_extensions-explainer.md
struct WebAppOriginAssociation {
  array<AssociatedWebApp> apps;
};

struct AssociatedWebApp {
  url.mojom.Url origin;
  string scope;
};

interface WebAppOriginAssociationParser {
  ParseWebAppOriginAssociation(string file_content) => (WebAppOriginAssociation association);
};
```

Web App Origin Association Parser Implementation

Set up utility process to parse web app origin association files (I04cec826) · Gerrit Code Review (googlesource.com)

We parse the file content into WebAppOriginAssociation here in the utility process, if possible, and return the data structure to the browser process.

```
namespace webapps {
```

```
class WebAppOriginAssociationParserImpl : public mojom::WebAppOriginAssociationParser {
 public:
  // web_app_origin_association::mojom::WebAppOriginAssociationParser:
  ParseWebAppOriginAssociation(const std::string& file_content,
ParseWebAppOriginAssociationsCallback callback);

};
}
```

WebAppOriginAssociationParserService

[Set up utility process to parse web app origin association files (I04cec826) · Gerrit Code Review (googlesource.com)]

Defines a function to launch the parser utility process.

```
namespace webapps {

mojo::Remote<mojom::WebAppOriginAssociationParser> LaunchWebAppOriginAssociationParser() {
  return content::ServiceProcessHost::Launch<mojom:: WebAppOriginAssociationParser >(
    content::ServiceProcessHost::Options()
      .WithDisplayName(
          IDS_WEB_APP_ORIGIN_ASSOCIATION_PARSE_DISPLAY_NAME)
      .Pass());
}

}
```

WebAppOriginAssociationManager

[Add Web App Origin Association Manager to fetch, parse, and validate associations (I43d26560) · Gerrit Code Review (googlesource.com)]

This class manages Web App Origin Association related work. It creates a Task for each batch of scope extensions to call `WebAppOriginAssociationFetcher` to make network requests for association files, and feeds the file content into `WebAppOriginAssociationParser`. Once the association files are parsed, it validates the associations. It handles starting a mojo remote connection to the utility process and

communicating with it, as well as task management to make sure batches of scope extensions are handled in order.

```
// web_app_origin_association_manager.cc
namespace web_app {

WebAppOriginAssociationManager::WebAppOriginAssociationManager() = default;

WebAppOriginAssociationManager::~WebAppOriginAssociationManager() = default;

const mojo::Remote<web_app::mojom::WebAppOriginAssociationParser>&
WebAppOriginAssociationManager::GetParser() {
 if (!parser_ || !parser_.is_bound()) {
   mojo::Remote<web_app::mojom::WebAppOriginAssociationParser> parser(
      LaunchWebAppOriginAssociationParser());

   parser_ = std::move(parser);
   parser_.reset_on_disconnect();
 }

 return parser_;
}

}
```

WebAppOriginAssociationFetcher

[Add a fetcher for web app origin association files (Iba5dd6ec) · Gerrit Code Review (googlesource.com)]

This class makes network requests for Web App Origin Association files in the browser process. It will use `net` libraries such as `network::SimpleURLLoader` to do so.

When To Fetch Association Files

There are several scenarios where we would need to fetch the web app origin association files and validate the information.

1. When we install a PWA, so that we can store the associated app info in WebAppDatabase.
2. When we update a PWA (i.e. manifest is updated), so that we can update the relevant info in WebAppDatabase.
3. When a user opens the Application pane (or some other appropriate tool) in DevTools against a PWA page, we would want to fetch, parse, and validate the associated app information and display it to the user.

On PWA Installation

Get web app origin associations at app install (If59356fc) · Gerrit Code Review (googlesource.com)

Web app origin association info should be fetched post-installation, since it won't be used until after app installation. To minimize the impact on web app installation user experience, we will start fetching and processing web app origin association files after app installation is finalized, as part of an OsInstallHook. Origin association fetching is done asynchronously to ensure the next steps of installation are not blocked.

We will check the scope extension information and if necessary, launch a utility process to fetch the appropriate association files and validate the information. The validated info will be persisted to WebAppDatabase by updating the list of app scope extensions.

On Manifest Update

Update URL handlers at manifest update (Iba427da0) · Gerrit Code Review (googlesource.com)

Currently, manifest updates are throttled to once a day or longer. When we navigate to an installed PWA page, we will check if an update should happen. If so, a manifest update task is created to fetch the manifest and check the new content against what's been installed. If updateable changes are detected, we will update the app after all app windows are closed. The call stack:

```
ManifestUpdateTask::IsUpdateNeededForManifest
ManifestUpdateTask::OnDidGetInstallableData
InstallableManager::GetData
ManifestUpdateTask::DidFinishLoad
ManifestUpdateManager::MaybeUpdate
```

WebAppTabHelper::DidFinishNavigation

When checking if manifest update is needed, we will check if scope extensions have changed, and schedule an update if they have. If a manifest update isn't needed, we will check if scope extensions exist. If so, associations need to be fetched and validated.

Once we have the updated associations, we need to update WebAppDatabase accordingly. If a scope extension changed (i.e. scope changed), it needs explicit user permission again to be enabled. Any scope extensions that don't exist anymore will be removed from WebAppDatabase. If a scope extension already exists in WebAppDatabase, we will keep the user permission unchanged.

Since associations are updated as part of the manifest update, the update frequency is once a day as well.

On DevTools

The Application Panel currently has support for PWAs through CDP methods under the Page domain: `Page.getAppManifest`, `Page.getInstallabilityErrors`, `Page.getManifestIcons`. A new CDP method can be added to fetch web app origin association files.