Isolate Browser Session Tabs

Daniel Lammens

March 5th, 2022

1 Background

1.1 Literature Review

Starting this task, I wanted to find prior examples of tab session isolation extensions. By doing so I could better understand the core problem of isolating tabs, and see how others had attempted to solve it. I found several examples, including Mozilla's Multi-Account Containers, the open-source Temporary Containers, and the Chrome extension Session Box. All worked off the logic of separating tabs into memory-groupings, where local storage, session storage, and cookies would all be separated based on which container the user was currently operating in. There was some difference between them, but I was told to focus on a Chrome-specific implementation, so I examined the Chrome extensions more intently. But I still didn't really understand *how* they were doing this, so I had to do some more digging.

1.2 Storage Research

After doing some brushing up on Authorization, I knew that no matter the authorization method (traditional cookie vs token-based), the important piece of authorization was always going to be stored in the cookies. So the key issue of this task was separating cookies by which tab they originated from. I had explored some other options: storing the authorization token in sessionStorage, modifying the URL header independently for each tab, or changing the file Chrome uses to save its cookies into a folder, with independent files for each tab, but as I delved further, these seemed prone to issues and I scrapped them in favor of the popular container solution.

2 Finding Answers

Knowing that I wanted to separate cookie storage, I delved into the source code of one of the open-source options, Temporary Containers, to check how they did it. The documentation wasn't excellent, so it took a bit of digging to understand the core functionality of the container system. As I understood it, the system relies upon matching two properties: the tab's extended cookieStoreID property, and the cookie's storeID property. For a Chrome solution, this requires the use of the chrome.cookies API. In the chrome.cookies API, you can see the Cookie Type, which has the additional property storeID, the ID of the Cookie Store to which this Cookie belongs. This attribute separates this API's cookies from normal document.cookie, which does not contain this property. You can also see the CookieStore Type, which contains two properties, id (see above), and tabIds, an array of all of the browser tabs that share that cookie store.

2.1 Cookie Stores

So now I had the answer, I should create a new cookie store for each container; whether that be for each tab or a list of 'profiles'. But I didn't know how to create a new cookie store. The Type 'Cookie Store' and the method getAllCookieStores() were listed in the documentation, but there wasn't an equivalent to cookie.set() for adding cookie stores. In the documentation, Chrome says "An incognito mode window, for instance, uses a separate cookie store from a non-incognito window.", so I went looking for how Incognito creates its store. This turned out to be a goose chase, as documentation was unhelpful, and questions were more geared to the functionality of Incognito than its implementation. Some answers referenced the potential solution I had alluded to earlier, that Incognito separated its cookies by creating a separate file location with which to save its cookies (and deleting said file upon close), but some answers refuted this as its implementation, and with no good official documentation, I could not find the true answer. I decided that, for the purposes of this write-up, I would assume that there is a way to create a Cookie Store in local storage, which I will treat as

```
new CookieStore(id, tabIds)
```

3 Implementation

After having done this research, I set about writing some pieces of code that I thought would be critical to a system of Isolated Browser Tabs. I made them as functional as possible, but in some places I had to give assumptions for parts that would not be efficient to code, given the time constraint. I have noted these in comments.

```
import {v4 as uuidv4} from 'uuid';

/**
 * When a new tab is created, create a new CookieStore, and assign
    that tab
 * the new Cookie Store's ID in its .cookieStoreID property.
 * Could be modified to group tabs in Cookie Stores based on
    container.
 */
chrome.tabs.onCreated.addListener(
    function(tab){
        const newStore = new CookieStore(uuidv4(), [tab.id])
        tab.cookieStoreID = newStore.id
    }
  );

// The following functions are based on parts of the Temporary
   Containers extension

/**
 * Take the list of cookies coming from this domain, and add the .
    storeId
 * property, giving it the storeId of the tab from which it was sent
 *
 * @param {Tab} tab - The tab from which this cookie is being sent
 */
async function setAndAdd(tab){
    // thisDomainCookies is an assumed array of cookies that match
       this domain
    for(let cookie of thisDomainCookies) {
        /* Create a cookie from the original cookie data, but
```

```
            with the added storeId property that matches this tab */
        const newCookie = {
            domain: cookie.domain,
            expirationDate: cookie.expirationDate,
            httpOnly: cookie.httpOnly,
            name: cookie.name,
            path: cookie.path,
            secure: cookie.secure ,
            url: cookie.url,
            value: cookie.value,
            sameSite: cookie.sameSite,
            storeId: tab.cookieStoreId,
        };
        await chrome.cookies.set(newCookie);
    }
}

/**
 * Check whether the given cookie exists within the given tab's
    Cookie Store
 *
 * @param {Cookie} cookie
 * @param {chrome.webRequest._OnBeforeSendHeadersDetails} request -
    Server requesting cookies
 * @param {Tab} tab
 * @returns Details of cookie in this tab's cookie store, with
    @param cookie's name.
 *          If no such cookie found, returns null.
 */
async function checkCookie(cookie, request, tab){
    const cookieAllowed = await chrome.cookies.get({
        name: cookie.name,
        url: request.url,
        storeId: tab.cookieStoreId,
      });
    return cookieAllowed;
}
```

```
/**
 * Add cookies from given tab to Map that may be turned into a
    cookie header for given request
 *
 * @param {chrome.webRequest._OnBeforeSendHeadersDetails} request -
    Server requesting cookies
 * @param {Tab} tab - Tab checking which cookies exist in its Cookie
     Store
 * @returns Map of cookies that are allowed to be sent to the server
     who gave the request
 */
async function addRequestCookies(request, tab){
    const newCookiesHeader = new Map();
    // allCookies is an assumed array of all cookies
    for(let cookie of allCookies) {
        let allowedCookie = checkCookie(cookie, request, tab);
        if(allowedCookie){
            newCookiesHeader.set(allowedCookie.name, allowedCookie.
               value);
        }
    }
    return newCookiesHeader;
}
```

Note that this implementation is only possible as Chrome extensions may use the chrome.tabs, chrome.cookies, and chrome.webRequest web APIs.

3.1 JSON Files

package.json

```
{
    "name": "Tab Task",
    "description": "",
    "authors": "Daniel Lammens",
    "version": "1.0.0",
    "main": "background.js",
    "dependencies": {
```

```
        "uuid": "8.3.2"}
}
```

manifest.json

```
{
    "name": "Tab Task",
    "manifest_version": 3,
    "version": "1.0",
    "permissions": ["cookies", "tabs", "webRequest"],
    "host_permissions": ["<all_urls>"],
    "background": {
        "scripts": ["background.js"]
    }
}
```

4 Conclusion

This was an interesting task. I had to learn more about extensions and their privileges, try a bunch of solutions that lead to dead ends, and discover that some things I was trying to implement from scratch were already part of native chrome WebAPIs. I'm glad that they were, though, I think this task would've been improbable within the time frame without the cookies and tabs APIs. There was a lot of documentation and stack-overflow searching that is not apparent in this write-up, as much of it lead to unhelpful places, but that is a part of development. There are definitely a lot of things that I didn't figure out, including how to actually create a new Cookie Store "Incognito-style", but given the scope of this task, I'm sure I'll be able to find those out in the future. As well, as I was told to focus on a Chrome-specific solution, I developed in that direction, and it would take time to make any of these bits of code compatible with other browsers. However, I think the idea behind the solution is a fruitful one, and given more person-hours, I believe it could be a valuable extension, as seen by the fully-functional extensions mentioned in my Literature Review.