Brows Personæ

a web browser that inhibits tracking and allows multiple sign-ins by sandboxing the user's identity between different websites

from **Talus Baddley**

Abstract

There is a privacy problem on the web where an advertising or data mining firm can use participating websites to track individual users' browsing habits as they visit different sites. This is largely due to the fact that every mainstream web browser currently stores all of the user's identifying information in a single, unified pool.

I propose a web browser, called Brows Personæ, that identifies for a given tab the site that the user intended to visit; and only makes available the identifying information which was set when intending to visit the same site. I discuss the insufficiencies of the current cookie system, and present a concrete, simple method of working around those insufficiencies, grounded in human interface design to minimize the added complexity of using this method. I then discuss exactly how I implemented and tested these features.

Contents

In this PDF, the Table of Contents and blue-colored links are clickable.
Text hard to read? Be sure to download and read this outside of Canvas!

Introduction

Privacy online is in a dismal state, with rampant spying being perpetrated by the advertisers and big data mills, the Googles and the Facebooks of the internet. Where you go and what you do online is being tracked by these sorts of companies via embedded ads and buttons, and no one really knows what they do with this information—at best, it's used for targeted marketing. It's a well-known axiom in the computer industry, though not so much in the general public, that whenever a product is free, the product is actually *you*. And more often than not, it's your own activity data that's most valuable.

The history of this particular privacy problem is a long one. It starts with the simple original design of HyperText Transfer Protocol: This protocol was not designed, at first, to handle state, but as the web saw the need for login and payment information, cookies were added to make stateful browsing possible. The only security policy put in place regarding cookies was that they could be accessed only by the site that set them. This measure has not been really expanded upon since; and it may have been all that was necessary, if it were not for resources from different sites being loaded on the same page, and JavaScripts to connect information between these resources. What we see most commonly now is an iFrame (embedded webpage), paired with a small script, giving an external website within the frame access not only to its own cookies identifying the user, but also information from the enclosing page. This is used by advertisement companies and big data services, chief among them Google and Facebook, who each have massive collections of sites embedding their frames (in the form of ads, like-buttons, etc.), and massive numbers of logged-in users who can be paired with those metrics.

The security problem with cookies is not that they're available to embedded pages and resources. The problem is that a single cookie for site A is available when

- visiting site A,
- visiting site B with an ad from site A, and
- visiting site C with an ad from site A.

This means a single user's identity (within that one cookie) can be tracked between completely different and unrelated sites (B and C), so long as they participate with site A.

Brows Personæ does not attempt to change this mechanism for cookies—after all, many sites rely on it for completely legitimate purposes. What I instead propose is a more sophisticated way of handling cookies, based on which site the user originally *intended* to visit:

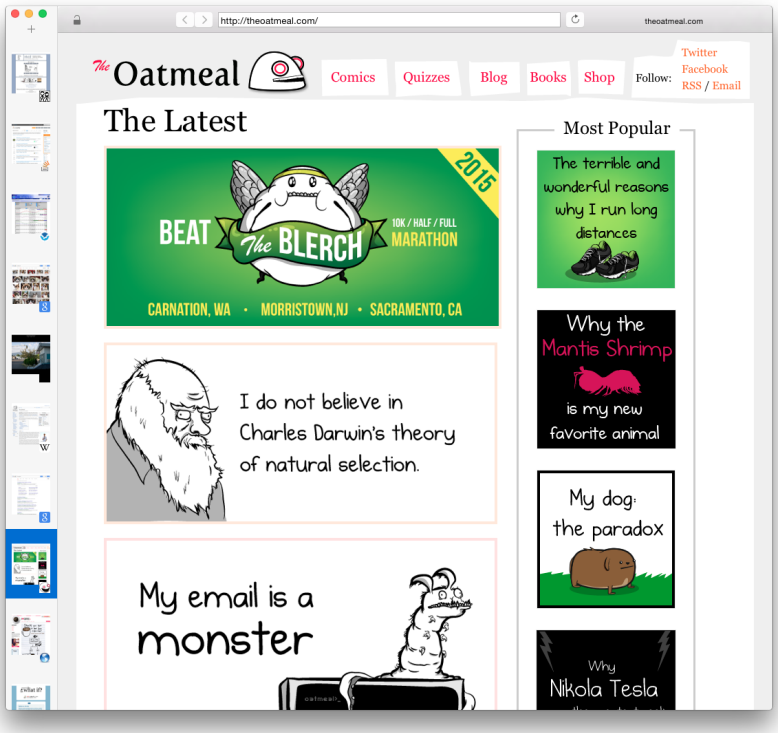
- When visiting site A, a cookie A_A is set.
- When visiting site B, an ad from site A can set and access cookie A_B, but has no knowledge of cookie A_A.
- When visiting site C, an ad from site A can set and access cookie A_C, but has no knowledge of cookies A_A or A_B.

Each site the user intended to visit is paired with a persona, which maintains a distinct and isolated set of cookies, collected from any site the user may directly or indirectly visit while using the persona. This isolation keeps the user's identity strictly separated between each persona, just as if they were using a different computer to visit each site.

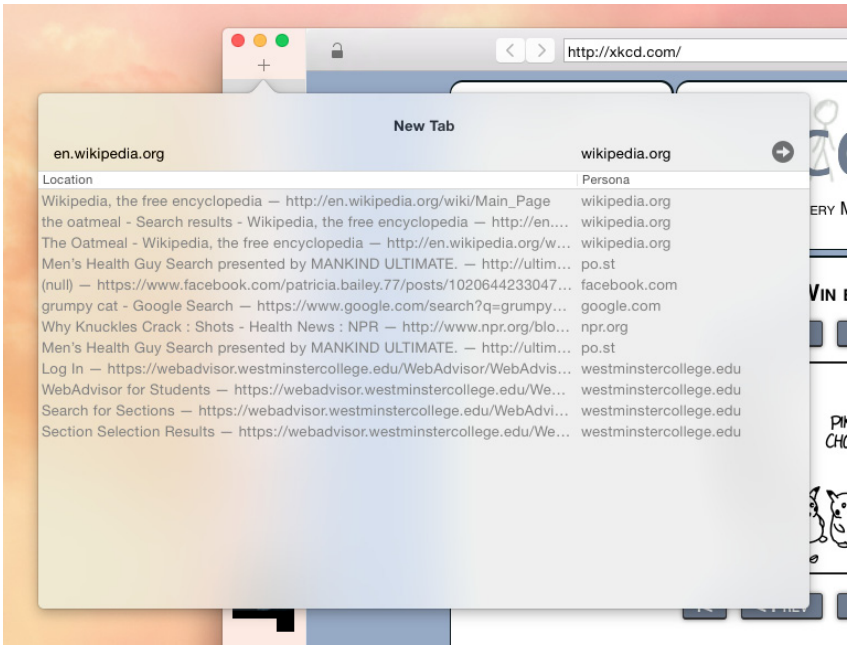
One challenge in this approach is determining where the user intends to go—or, rather, allowing them to tell the application in the easiest way possible. Nobody wants to use a browser that's more difficult than the one they're used to.

Human Interface Description

The root element of the browser is the Brows Window, which contains any number of Brows Tabs. The tabs are arranged along the left-hand side of the window, rather than the top, and they show a thumbnail of the page. The list supports any number of tabs without shrinking them down, by simply scrolling the list; and the thumbnails make the tabs easier to identify and switch between.



When making a new tab, rather than just opening it blank or to a home-page, Brows Personæ opens a little pop-up panel asking the user to pick a location. They can either select from a list of bookmarks or type a location. (By "location" I mean either a URL or Google search.) As they type in the location, they implicitly indicate the site they intend to visit, and so the panel will infer the persona that the new tab will use. They can then hit return to go, or can optionally change the persona that they'll use first.



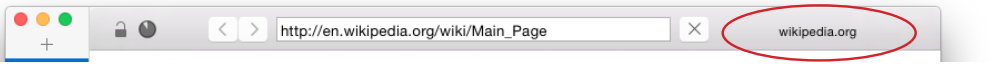
Personæ are identified by name, and normally a persona is named after the base host of the URL the user entered in the new tab panel. For instance,

- Visiting `http://foo.com/bar/baz` will default to persona `foo.com`.
- Visiting `http://quux.bar.co.uk/home` will default to persona `bar.co.uk`.

Brows Personæ uses Mozilla's Public Suffix List to determine the basic registrable part of the domain name—so, in the second example, "quux" is not included in the persona name, because "bar" is the root of private ownership under the public suffix `co.uk`.

The user may, however, choose any string as a persona name when making a tab. The default is provided as an easy, intuitive shortcut. But the user may visit the same site (say, facebook.com) under distinct personæ ("facebook.com-1" and "facebook.com-2," for instance), allowing for simultaneous sign-ins under different accounts.

Once a tab has been created, its persona is fixed. Navigating a website often results in visiting links to external sites, and in doing so, information can be passed along (most simply, in the URL itself). Switching personæ could cause this passed info to lose its context, and some sites would not function. Instead, when visiting multiple sites in a tab, Brows Personæ uses the originally-intended persona for all requests within the tab. This makes for a simpler browsing experience, as the user knows that everything they do within a tab happens under the single identity that's displayed at the top.



Implementation

I built Brows Personæ as a Mac application, for a few reasons. First, it needs to be smooth and consistent to use, and reasonably attractive. Second, there was a limited amout of time to get it up and working. As a consequence of these two, it needed to be platform-specific. And third, WebKit (the webpage renderer and platform) integrates very well with Mac OS X. I had significant experience doing Cocoa development, so creating a Mac app was the most sensible choice.

Project Structure

Half of the effort building the browser was devoted to making a well-functioning application. Near the top level, the Brows Windows and Brows Tabs each have their own controllers that handle most of the application logic. A Brows Window has a Make-Tab Controller that handles the new-tab panel. And a Brows Tab has a Brows Persona and, of course, a WebView provided by WebKit.

Cookie Handling and WebKit

There are three pillars of security that Brows Personæ works on: HTTP cookie handling, JavaScript cookie handling, and private browsing mode. I'll describe each:

HTTP Cookie Handling

The web view I use is a subclass of WebKit's, based on `IGIsolatedCookieWebView`, which modifies the behavior of the WebView to directly handle cookies that the view deals with in HTTP headers, rather than defaulting to the standard shared cookie store. It does this by way of delegate callbacks that are part of the standard WebView API. As the view makes requests, the delegate attaches its own cookies; and as the server sends responses, the delegate strips incoming cookies of and stores them itself. This subclass, as originally written by Isaac Greenspan, stores cookies in a transitory private array. I extended this behavior to instead pass cookie-getting and -setting off to the persona associated with the tab.

Each Brows Persona object offers methods for setting and retrieving cookies in a number of ways. It stores cookies by maintaining its own small SQLite database, located at `~container/Library/Application Support/Brows Personæ/Personæ/`*`name`*`.browspersona`. This database contains tables for cookies, browsing sessions, and history as the user navigates within tabs associated with the persona. There are several reasons I elected to store cookies in a database rather than a flat file:

- SQLite guarantees the integrity of the database even with concurrent writers. If multiple instances of Brows Personæ—or even other tools or scripts—were to use a persona at the same time, there would be no special race considerations. The database also immediately synchronizes cookies among all these parties.
- SQLite is a highly standardized and well-documented format, and allows anybody to freely inspect the table structure and cookie/history data.

- Because of foreign-key constraints and cascading deletions, the app can, for instance, periodically trim old session-cookies by simply deleting the session they point to.

JavaScript Cookie Handling

WebViews provide a standard API for customizing HTTP requests and responses, and we used that to our advantage in the previous section, attaching and stripping cookies from those headers. However, WebKit provides no similar API for handling requests for cookies that come from JavaScript on the page while it's running. What I discovered was that the Mac implementation of WebKit is hard-wired to a class in the standard library called `NSHTTPCookieStorage`. This class maintains a shared singleton that keeps all cookies synchronized throughout the application. Thus, I needed to modify WebKit itself to break these hard connections and use my own Brows Persona objects instead. Fortunately, it's open-source.

I had to first modify WebKit to be embedded in an application, rather than to be installed in the OS as it's designed. I also had to update it to use the same code signature that I use with Brows Personæ itself. Before doing the modification I also had to parse out the structure of the massive project and develop a strategy for "lowering in" an object of my own into the befuddling morass of C++ and Objective-C++ code that makes up WebKit, providing all the right headers and so on.

There is a `Page` object that's available almost anywhere in the WebView stack. I ended up adding an instance variable to this class to hold my Brows Persona object directly. I also added some setter and accessor methods higher up in the stack that I could use to insert the persona as necessary.

I then added an Objective-C/C++ bridge to handle cookie requests from the JavaScript handlers—and finally modified the JavaScript handlers to use this bridge instead of `NSHTTPCookieStorage`.

All of these changes are available for inspection on GitHub.

Private Browsing Mode

Similar to Incognito mode in Google Chrome, Private Browsing mode is not very useful on its own. However, in WebKit it does enable a few nifty features that help protect the user's privacy. For instance, it keeps shared history from coloring certain links as visited, and it de-tunes the resource caching policies—each of which could otherwise be used to compromise the user's anonymity. Enabling this mode for all Web Views in the app was an easy way to opt in to these very useful privacy features.

Results

I set forth two criteria for success of this project: that it be an elegant and functional web browser that's easy to use, and that it properly isolate the user's identity between personæ. It is successful on both counts.

Easy-to-Use Application

Brows Personæ is missing a number of features that users expect, and so it's not yet ready for public release. But it is a functional, stable application that displays nearly all webpages as correctly as Safari does. And since it infers the user's intent as they open tabs, and does not require any extra input to select a persona, I'd judge that it's as easy to use as any browser.

Properly Isolates Identities

There are a number of sneaky ways to hide cookies not just in standard cookie storage but in other, seemingly-unrelated browser stores such as history items. A JavaScript library called Evercookie attempts to combine all such known methods to save cookies that are virtually irrevocable. I decided to use this as the benchmark to detect whether any user information could leak from one persona to another.

If you visit `http://samy.pl/evercookie` in a tab under one persona (e.g. "samy.pl-1") and set a random cookie, then visit the same page in a new tab under a different persona ("samy.pl-2"), you will find that the second page is unable to find the cookie that was set in the first page. So I believe that for all practical purposes, no usable information leaks between personæ, and the browser properly isolates the different identities.

Conclusion

Brows Personæ is a promising web browser that, for the scope of this project, fulfills its goals.

I fully intend to make a number of enhancements to the usability of the interface—a downloads window, tab rearranging, etc—and do a beta release in the near future. If it becomes popular enough, I will continue to guide its development, but pass off maintenance to the open source community.

Ultimately, the best outcome would be for the mainstream browsers (Chrome, Firefox, Safari, …) to adopt the simple persona-based privacy measures I've set forth. No outcome would be more successful than them adopting this system; if Brows Personæ needs not be anything more than a proof-of-concept, nothing would make me happier.