Extending the Same Origin Policy with Origin Attributes

Tanvi Vyas, Andrea Marchesini and Christoph Kerschbaumer
Mozilla Corporation, Mountain View, U.S.A.
{*tanvi, baku, ckerschb*}*@mozilla.com*

Keywords: Browser Privacy, Browser Security, Origin Isolation, Cookies, Web Tracking.

Abstract: The Same Origin Policy (SOP) builds the foundation of the current web security model. As the web evolves, numerous new specifications propose extensions to the SOP in order to improve site security or improve user privacy. Site operators benefit from an extension to the SOP because it allows sites to partition their physical origin space into many different contexts, each representing their own abstract origin. Users benefit from an extension to the SOP because it allows users to separate user data for privacy purposes and enables richer browsing experiences. Implementing any of these new features requires tremendous engineering effort for browser vendors and entails the risk of introducing new privacy concerning vulnerabilities for end users. Instead of spending considerable engineering effort to patch the browser for every new specification that proposes to extend the SOP, we re-design a web browsers architecture and build Origin Attributes directly into a browsers rendering engine. Our implementation allows any specification or web technology to integrate into Origin Attributes with minimal engineering effort and reduces the risk of jeopardizing an end user's security or privacy.

1 MOTIVATION

Web applications rely on cookies and other data storage mechanisms to remember user state in the otherwise stateless HTTP protocol. Remembering user state allows sites to provide richer user experiences and keeps users from having to continuously remind sites about their state and preferences. However, such storage mechanisms not only enable websites to remember user preferences, but also enable integrated third party content to track users across the web.

As the web continuously expands, the boundaries between sites become increasingly blurred. Almost every modern website includes content from social media sites and/or includes content that allows the site operator to analyze user behavior. Mashups integrate content from various sources to provide new services. Ultimately all sites include numerous third party resources which can track users across sites (Englehardt and Narayanan, 2016; Yu et al., 2016; Englehardt et al., 2015; Acar et al., 2014; Tran et al., 2012). Unfortunately, users have little control over which third party resources load within the sites they visit and hence have limited control over tracking.

The current web and browser architecture enables tracking because cookies and site data are stored together within the same data storage (a Cookie Jar) that separates data solely based on the SOP. Browser vendors attempt to provide users relief from tracking with features like Private Browsing Mode (Mozilla, 2009), Tor's First Party Isolation technique (Perry et al., 2016), and Firefox Containers (Mozilla, 2016). Each of these features extends the SOP by additional labels to further segregate content on the same origin. Site operators have also expressed their desire to further refine the SOP with features like Suborigins (Weinberger and Akhawe, 2016) and Isolate-Me (Stark et al., 2016). All of these features require an extension to the SOP to isolate data by introducing additional labels.

Developing each feature individually within Firefox proved to be complex and required thousands of lines of feature specific code. Moreover, features developed individually within Firefox were incomplete and some browser components even had to be disabled in order to achieve the required isolation. Worse, in some cases the browser was not able to fully isolate data. Instead of continuing to integrate these new features into the browser one at a time, in an ad-hoc and fallible manner, we restructured Firefox' isolation system using a technique we call Origin Attributes. Origin Attributes allow each feature to add additional labels to the origin, which are then automatically enforced together with the SOP when sites attempt to store and retrieve data from a Cookie Jar. Additionally, if a vulnerability is discovered and fixed for a particular feature that leverages Origin Attributes, all consumers of Origin Attributes can benefit from the fix to achieve

Vyas, T., Marchesini, A. and Kerschbaumer, C.
Extending the Same Origin Policy with Origin Attributes.
DOI: 10.5220/0006210404640473
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 464-473
ISBN: 978-989-758-209-7

the desired isolation.

The rest of this paper is organized as follows. We first summarize web architecture fundamentals (Section 2), providing background information on the Cookie Jar, the SOP and how current web architecture allows tracking sites to track users across the web. We then contribute the following:

- We provide design and implementation details for integrating Origin Attributes into Firefox (v.52.0) (Section 3).
- We present how Firefox Containers (Section 4) and First Party Isolation (Section 5) leverage the presented Origin Attributes.
- We evaluate the Engineering Effort for Origin Attributes, Firefox Containers, and First Party Isolation, and then survey additional consumers of Origin Attributes (Section 6).

2 BACKGROUND ON CURRENT WEB ARCHITECTURE

Before we can introduce Origin Attributes, we first establish semantics of the web and highlight current web fundamentals, starting with the Cookie Jar.

2.1 Cookie Jar

We refer to the Cookie Jar as a collection of HTTP Cookies, Web Storages, and the contents of the Browser Cache.

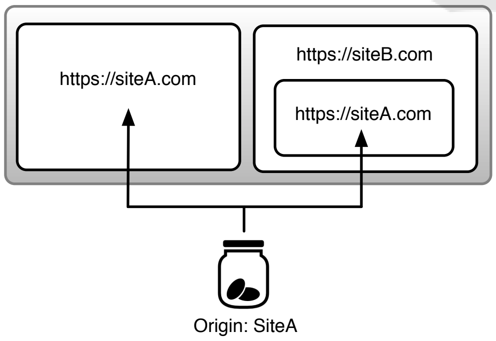

Figure 1: Cookie Jar in current browser model.

As illustrated in Figure 1, current browser architecture labels data within a Cookie Jar by its origin. When accessing an origin's data from the Cookie Jar, the browser does not distinguish whether that origin is loaded at the top level or within an iframe.

HTTP Cookies: An HTTP Cookie (Barth, 2011) defines a small piece of data that a server sends to the user's browser. Cookies define a mechanism to remember stateful information in the otherwise stateless HTTP protocol and typically consist of a *name*, *value*, and zero or more *attributes*. The browser stores the cookie locally on the user's device and sends that piece of data back with subsequent requests to the same origin.

Web Storages: In addition to HTTP Cookies, the browser also offers sites other local data storage mechanisms, including: *LocalStorage*, which allows a site to store string values on the client's browser that exist beyond the duration of the session and that can go beyond the maximum size constraints of the HTTP Cookie. *IndexedDB*, which provides an API for client-side storage of larger amounts of structured data. In contrast to the afore mentioned LocalStorage, the API of IndexedDB enables high performance searches of the stored data. *Cache API*, which allows a site to implement its own persistent cache.

Browser Cache: The browser stores certain data keyed by origin in its cache so that it is readily available if the data is requested again in the future. Numerous sub systems rely on that cache, for example: the *Network cache*, which stores HTTP responses so the browser does not have to perform expensive network reloads of already downloaded resources, and the *Image cache*, which avoids redundant decoding of images if the image has already been decoded.

2.2 Same Origin Policy

Table 1: Performing SOP check against `http://siteA.com/page.html`.

URL: `http://siteA.com/page.html`	*SOP*
`http://siteA.com/other.html`	✓
`http://siteA.com/dir/other.html`	✓
`https://siteA.com/secure.html`	X
`http://siteA.com:81/etc.html`	X
`http://a.siteA.com/other.html`	X

All web browsers isolate Cookie Jar data based on the **Same Origin Policy** (SOP) (Barth et al., 2009). The SOP defines an origin as a combination of scheme, host and port number. Not only does the SOP prevent malicious script on one page from obtaining access to sensitive data on another web page, the SOP also defines what data from the Cookie Jar can be accessed by a site.

Table 1 illustrates the results of performing a SOP check against the URL `http://siteA.com/page.html`. As shown in Table 1, the SOP considers two resources to be identical if the domain name, the application layer protocol, and the port number are identical. Hence, the browser considers the URL `http://siteA.com/page.html` to be same origin with the URLs on Line 1 and 2, but considers the URLs listed on line 3, 4 and 5 to be cross origin based on a different protocol, different port, and different host respectively.

2.3 User Tracking

Even though the browser adheres to the guidelines of the SOP and restricts Cookie Jar data, current web architecture allows tracking sites to gain information about a user's behavior across multiple independent sites. The browser allows sites to embed third party resources from various domains so that sites can create new services and provide a rich web experience for the end user. Tracking sites can use the same embedding mechanism to trace users across the web. To illustrate the problem, consider the following example:

A user visits `https://siteA.com` and siteA requests the browser to store a cookie (see Figure 1). This cookie will then be sent back to siteA whenever the user visits the site. When the same user visits `https://siteB.com`, the cookies from siteA will not be sent since siteB is a different origin. However, if `https://siteB.com` embeds `https://siteA.com`, siteA's cookies will be sent with the subrequest for siteA along with a referrer from siteB. Now siteA has gained sensitive information and knows that the user (as identified by their cookies) has visited siteB.

3 ADDING ORIGIN ATTRIBUTES

Implementing features that extend the SOP have proven complicated and error prone within web browsers. Since the SOP provides a first line defense against cookie stealing, accessing unauthorized storage APIs, and inspecting elements within a browsers cache, one can imagine that every implementation error in extending the SOP has privacy and security implications for the end user. Further, all new features proposing to extend the SOP (described in Sections 4, 5, and 6.2) have to overcome the same obstacles and patch the same code throughout the codebase to plug their feature into Firefox.

To provide an infrastructure for features extending the SOP, we implemented Origin Attributes within Firefox (v.52.0). Origin Attributes provide a framework that any new feature that extends the SOP can build on top of. Before we can explain the integration of Origin Attributes into Firefox, we first establish common terminology about Firefox internals.

Terminology: Firefox relies on the concept of a Principal for performing security checks. A Principal represents a browsing security context compromised of the scheme, host, and port of an origin. To evaluate whether two contexts are same origin, Firefox compares the Principals of the two contexts.

For example, the web page `https://siteB.com` embeds a frame from `https://siteA.com` (see Figure 1). The siteA subdocument may request access to the cookies of the top-level document. To evaluate whether to allow the frame access to the top-level cookies, Firefox compares the Principal of the frame with the Principal of the top-level page for equality. Since siteA and siteB have different origins, their Principals do not match and hence Firefox denies the frame access to the cookies of the top-level page.

In general, Firefox distinguishes between three types of Principals:

- *Content Principal*: A Content Principal is associated with specific web content and reflects the origin of this content. For example, when loading `https://siteA.com` Firefox generates a Principal of `https://siteA.com` and will use that Principal to perform SOP checks.
- *Null Principal*: The Null Principal is a unique Principal that cannot be accessed by anything other than itself and system code. The Null Principal uses a custom scheme and host, e.g. `moz-nullprincipal:{0bceda9f-...}`, where the host is represented as a 128-bit universally unique identifier. Because of the unique ID, a Null Principal represents a resource that is only same origin with itself. Null Principals are used for iframe sandbox loads and `data:` URLs loaded in the top-level page.
- *System Principal*: The System Principal is associated with network loads that the browser (the system) initiates, as opposed to network loads that web content initiates.

3.1 Extending a Principal by Origin Attributes

Origin Attributes provide additional separation identifiers on Principals to determine whether two Principals reflect the same security context. If any of these additional labels differ between two Principals, the Principals represent a different security context. In more detail, since the introduction of Origin Attributes, two Principals are cross origin if any of the newly added Origin Attributes differ, even if scheme, host, port are the same as defined by the SOP.

Table 2: Performing SOP check against `http://siteA.com/page.html{0,'siteA.com',0,''}`, where `{0,'siteA.com',0,''}` reflects `{mUserContextId, mFirstPartyDomain, mPrivateBrowsingId, mAddonId}`.

URL: `http://siteA.com/page.html{0,'siteA.com',0,''}`	*SOP*	*Reason*
`http://siteA.com/other.html{0,'siteA.com',0,''}`	✓	
`http://siteA.com/other.html{1,'siteA.com',0,''}`	X	Different user context Id
`http://siteA.com/other.html{0,'siteB.com',0,''}`	X	Different first party domain
`http://siteA.com/other.html{0,'siteA.com',1,''}`	X	Different private browsing Id
`http://siteA.com/other.html{0,'siteA.com',0,'1'}`	X	Different addon Id
`http://siteA.com/dir/other.html{0,'siteA.com',0,''}`	✓	
`https://siteA.com/secure.html{0,'siteA.com',0,''}`	X	Different protocol
`http://siteA.com:81/etc.html{0,'siteA.com',0,''}`	X	Different port
`http://a.siteA.com/other.html{0,'siteA.com',0,''}`	X	Different host

```
struct OriginAttributes {
  uint32_t mUserContextId;
  nsString mFirstPartyDomain;
  uint32_t mPrivateBrowsingId;
  nsString mAddonId;
};
```

Listing 1: `Origin Attributes`-dictionary within Firefox.

Our newly added *OriginAttributes* structure currently consists of four dictionary entries which extend the scope of an origin. As new features develop and the need for new extensions arise, additional entries can easily be added. Each of the current entries is used for a feature that extends the SOP:

- *mUserContextId* is used for the Containers feature, as described in Section 4.
- *mFirstPartyDomain* is used for the First Party Isolation technique, as described in Section 5.
- *mPrivateBrowsingId* indicates whether Private Browsing is enabled, as described in Section 6.2.
- *mAddonId* is used to identify Web Extensions and their web content. For historical purposes, this attribute is also used by the Safe Browsing feature as described in Section 6.2

```
class Principal {
  public:
    bool isSameOrigin(Principal&
        aPrincipal);
  private:
    URI mCodebase;
    OriginAttributes
        mOriginAttributes;
};
```

Listing 2: `Origin Attributes`-attached to each Principal within Firefox.

As illustrated above, each Principal not only consists of a `URI mCodeBase` which reflects an origin defined as scheme, host and port, but now also consists of the newly added `OriginAttributes` that hold additional origin information.

Please note that Firefox's data structure `URI` predates the introduction of Origin Attributes. Since we did not want to pollute that data structure by extending it with additional labels, we decided to introduce and store `mOriginAttributes` as a separate member variable within a Principal. Even though a different implementation of Origin Attributes could use a different design approach, our decision to keep those two data structures separate is purely for historical reasons.

3.2 Extending the Same Origin Policy with Origin Attributes

In addition to extending the Principal class with a new `OriginAttributes` member variable, the member function `isSameOrigin(Principal& aPrincipal)` has also changed. The Principal passed into `isSameOrigin()` now only returns true if scheme, host, port as well as all four origin labels in the `OriginAttributes` dictionary are identical. In more detail, while the SOP defines two resources to be same origin if <`scheme, host, port`> are identical, we extended the SOP within Firefox so two resources are only same origin if <`scheme, host, port, originattributes`> are identical.

Table 2 illustrates the results of performing an extended SOP check against the URL `http://siteA.com/page.html{0, 'siteA.com', 0, ''}`. As shown in Table 2, the extended SOP considers two resources to be identical if the domain name, the application layer protocol, port number, as well as all four labels in the origin attributes dictionary are identical. Hence, the browser considers the URL `http://siteA.com/page.html{0, 'siteA.com', 0, ''}` not to be same origin with `http://siteA.com/page.html{0, 'siteA.com', 0, '1'}`. Even though the scheme, host and port are identical, the `mAddonId` in that example is different and hence the extended SOP considers those two Principals to be of different origin. This extension of the SOP further limits the way data can be stored and retrieved from the Cookie Jar and constitutes the key element in our approach.

3.3 Accessing Cookies with Origin Attributes

Internally, Firefox uses two data structures for maintaining cookies. In order to persist cookies after termination of a session, Firefox writes cookies into an SQLite database which is stored in a users' *profile* using the name `cookies.sqlite`.

For looking up cookies at runtime, Firefox uses an in-memory hashtable. Before introducing Origin Attributes the key for storing a cookie within that hashtable was computed by performing `key(origin)`.

Since the introduction of Origin Attributes, Firefox creates the key for storing the cookie in the hashtable by performing `key(origin, originAttributes)`. Even though extending the origin by Origin Attributes consumes a few more cycles when generating the hash value for a cookie, the introduction of Origin Attributes does not impact the look up time for finding that cookie within the hashtable. With or without the introduction Origin Attributes, the look up time for finding a cookie within Firefox is $O(1)$.

3.4 Leveraging Origin Attributes

Origin Attributes provide the plumbing for building privacy policies that propose to extend the SOP. Following we present two features: Containers (Section 4) and First Party Isolation (Section 5). We highlight how both of these features leverage the presented Origin Attributes and build privacy policies on top of it.

4 USER-CONTROLLED CONTEXTS (CONTAINERS)

Containers are user-defined lightweight persistent contexts that isolate sites within a web browser. At an abstract level, the feature provides a mechanism for users to gain more control over the data websites can access, like HTTP Cookies, LocalStorage, and IndexedDB. To separate data websites can access, Containers insert a user-controlled label into the Cookie Jar using the presented Origin Attributes, which allows users to decide which state to use when interacting with a site.

The main motivations for providing a mechanism like Containers for end users are the following:

Tracking: Current web architecture permits tracking sites to track users across websites. Unfortunately, users have little control over this tracking, because end users can not influence third party content a website includes. The best option users currently have to protect themselves against tracking is to install ad blockers. The downside of using ad blockers is that such blockers trigger false positives and hence block valid content, downgrading a user's experience instead of just preventing the end user from being tracked. In addition, since ad blockers internally rely on a blacklist approach, the user has no guarantees that there are no unidentified trackers tracking them as they browse the web.

Context Separation: Users create multiple accounts on a single site for various purposes. For example, users who have both a work account and a personal account on a site want to be logged into both of their accounts simultaneously. Or, users who share devices want a secondary user to be able to login to a user account without requiring the primary user to log out. Current web and browser architecture does not support such use cases, because cookies and site data are stored together within the same data storage and hence remaining logged in into two accounts on the same site is technically not feasible.

4.1 Containers and Origin Attributes

The Containers implementation uses the introduced Origin Attributes to separate the different contexts in which a user browses the web. Each Container has a unique and never reused id, set in the attribute `mUserContextId` of Origin Attributes. By adding this additional layer of separation to data websites can access in the Cookie Jar, Containers allow users to define and then choose between different browsing contexts when visiting a site, thereby overcoming current web architecture shortcomings.

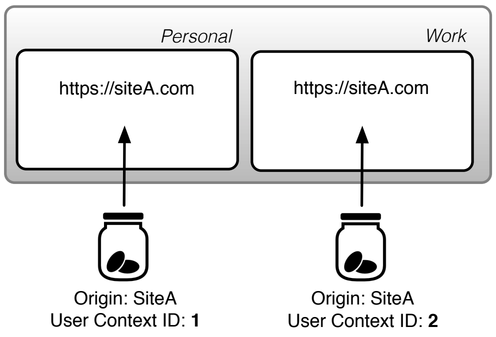

Figure 2: Cookie Jar Separation for Containers.

For example, assume a user visits and logs in to `https://siteA.com` in a defined *Personal* context and then again visits `https://siteA.com` in a defined *Work* context (see Figure 2). In contrast to regular browsing, siteA will not be able to identify and track the user in the *Work* context with the user's cookies from the *Personal* context. Containers prevent such tracking because Containers create two entries for `https://siteA.com` in the browser's Cookie Jar - one for `https://siteA.com` with an `mUserContextId` of *Personal (1)*, and one for `https://siteA.com` with an `mUserContextId` of *Work (2)*. Moreover, the user could login to a second account on `https://siteA.com` in the *Work* context. This enables users to be logged into multiple accounts on the same website simultaneously, even when the the site does not support concurrent session.

4.2 Container Isolation

Containers generally separate data accessible by web content while still sharing data that is only accessible to the user. Hence, Containers separate the following data and store them uniquely per browsing context:

- HTTP Cookies
- Web Storages (LocalStorage, IndexedDB, Cache API)
- Browser Cache (Network, Image cache)
- ServiceWorkers and SharedWorkers

The Containers feature still aims to support web browser fundamentals such as remembering a user's previous browsing data or providing a URL-bar that enables history navigation. These features enhance a user's browser experience without exposing data to web content. Hence, Containers make History, Bookmarks, the Password Manager, and Saved Form Data available to the user within every browsing context, while attempting to take extra precautions to ensure such data is not available to websites across contexts.

4.3 Preventing Tracking while Maintaining Usability

Tracking protection mechanisms follow an all or nothing paradigm and hence either allow all content to load or block all content. This blocking ratio causes sites to lose important functionality, or worse, sites may completely stop working in their intended way. We argue that users can still appreciate targeted suggestions in the right browsing context. Hence, Containers take a different approach to tracking; instead of blocking trackers completely, Containers allow sites to provide targeted suggestions in the appropriate contexts.

For example, imagine a user that restricts online shopping to a defined *Shopping* Container. While shopping, the user will see suggestions relevant to the user's shopping interests and find useful prodcuts. But while working in a defined *Work* Container, the user is not distracted with these products. Moreover, if users share their screen with their coworkers while in the *Work* Container, they will not have to worry about content they load in the browser leaking their personal shopping habits via targeted advertising.

Containers can further restrict tracking from websites that are known for being embedded across the web. For example, users want to stay logged into their social network site on their browser without enabling embedded social network buttons to track them across sites. Containers allow the user to remain logged in to their social network site using a specific context, and then browse the web in a different context where those social network credentials do not exist.

5 FIRST PARTY ISOLATION

The *First Party Isolation* technique isolates browsing contexts by the top-level domain (origin) the user visits to prevent embedded content from tracking users across sites. Current browser architecture sets and retrieves Cookie Jar data by the origin that set them. The downside of using just a single key for setting and retrieving such data is that such a mechanism enables third party content to track users across the web. Instead of using the third party resource's origin as a single key, the First Party Isolation technique relies on the origin of the third party resource in combination with the first party domain. Together those two domains form a double-key which the browser then enforces to grant or deny Cookie Jar access. So if siteA is embedded in siteB (top-level), siteA can only access Cookie Jar data that is double-keyed with both siteA.com and siteB.com. To achieve First Party Isolation, Firefox introduced the *mFirstPartyDomain* Origin Attribute which extends the origin and technically enables double-keying. [1]

5.1 First Party Isolation and Origin Attributes

When First Party Isolation, or double-keying, is enabled, Firefox will only grant a server access to Cookie Jar data when scheme, host, port, and the `mFirstPartyDomain` of the request match the stored

[1]Firefox users can enable this feature by setting the preference privacy.firstparty.isolate to true in `about:config`.

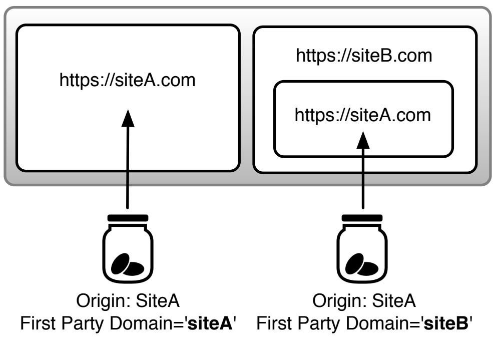

Figure 3: Cookie Jar separation for First Party Isolation.

origin. As illustrated in Figure 3, a user visits siteA and siteB in two separate windows. Additionally, siteB embeds content from siteA. Assume the user logs into siteA in the siteA window and siteA sets login cookies for that user. Firefox keys the login cookies for the origin `https://siteA.com` along with the first party domain in the URL-bar, in this case also siteA (`mFirstPartyDomain='siteA.com'`). The user then visits siteB in a different window where siteB includes an iframe to siteA. Firefox will grant embedded siteA access to cookies that are keyed by URI `https://siteA.com` and `mFirstPartyDomain='siteB.com'`, since siteB is the first party domain in the URL-bar. Since siteA's login cookies have an `mFirstPartyDomain='siteA.com'`, embedded siteA does not get access to them as the embedded content has a different mFirstPartyDomain. Without access to the user's login cookies, embedded siteA can not easily identify or track the user on siteB. In fact, siteA will not be able to identify or track the user by their login cookies on any site other than when siteA itself is the top-level domain.

5.2 First Party Isolation in the Tor Browser

We did not invent the concept of using a double-key as an access gate for cookies. The *Tor Browser* (The Tor Project, 2012), which consists of an *Extended Support Release* (ESR) version of Firefox, uses the concept of First Party Isolation by default within their browser (Perry et al., 2016).

Up to now, the Tor Project had to support, maintain and apply their First Isolation patches on top of Firefox ESR. One can imagine that rebasing patches that consist of thousands of line of code (see Section 6) can be quite time consuming. Especially since rebasing those patches to enable First Party Isolation within Tor is a privacy critical task. Any introduced error caused by maintaining these patches might reveal the identity of a Tor user. After all, Tor is designed to defend against network surveillance and to allow end users to browse the web without revealing their identity.

With the introduction of Origin Attributes, Firefox was able to integrate Tor's First Party Isolation technique directly into the browser. Starting with Firefox (v.52.0), the Tor browser can simply change a preference that enables First Party Isolation and will no longer need to support their own implementation of the feature.

5.3 Tracking Protection through First Party Isolation

Just like with Containers, (as described in Section 4), First Party Isolation does not outright block third party content or third party data access. Third party content can still load and provide sites with additional functionality. But, by using the double-keying mechanism, First Party Isolation prevents third parties from being able to track users across sites.

Note that while the Containers approach allows third parties to provide targeted suggestions across sites only while within a single browsing context, First Party Isolation allows third parties to provide targeted suggestions only while within a single first party.

5.4 Isolating Nested Browsing Contexts

First Party Isolation does not currently attempt to "triple-key" data when a site contains multiple iframes nested within one another. For example, assume the user has siteA open in a window. siteA embeds child siteB and child siteC. Now assume child siteB also embeds grandchild siteC. With First Party Isolation, child siteC and grandchild siteC can share data and communicate, since they are both double-keyed by origin `https://siteC.com` and *mFirstPartyDomain='siteA.com'*. But if we introduced a concept of "triple-keying", such that grandchild siteC had a *mFirstPartyDomain='siteA.com ^ siteB.com'*, for example, then we could further restrict data access. If new security or privacy threats arise that could be mitigated by triple-keying, the amount of effort to add triple-key support to Firefox within Origin Attributes would be trivial.

6 ENGINEERING EFFORT AND ADDITIONAL CONSUMERS

6.1 Engineering Effort

Origin Attributes: In order to integrate Origin Attributes into Firefox we have landed (at time of submission) 101 changesets with a total diff of 23,724 lines of code. Please note that all landed changesets are generated using `hg diff -p -U 8` which provides eight lines of context and shows the relevant function for the block.

Over a period of 18 months, three engineers worked full time on integrating Origin Attributes into Firefox and dozens of others provided feedback, guidance and reviewed the code. Worth mentioning is that Firefox is organized as modules, which means that one of the peers responsible for the code quality within a subsection of the code tree needs to review and accept the code before it can be merged into the codebase. Since this project modified code in all corners of the codebase, we had numerous discussions with reviewers from various parts of the codebase. We have invested approximately 6,000 man-hours to integrate Origin Attributes into Firefox.

Containers: To integrate Containers into the provided Origin Attributes, we only had to land 9 changesets with a total diff of 857 lines of code. We argue that integrating a new feature into the provided Origin Attributes requires minimal engineering effort.

In addition to the privacy relevant backend changes for supporting Containers, we landed 33 changesets with 6,954 lines of code to provide the necessary user interface elements for Containers. Even though providing user interface elements requires more engineering effort, we argue that adding elements to the interface is not privacy critical. Many of these changesets were added to enhance user experience of the Containers feature and are orthogonal to the backend modifications to integrate Containers into Origin Attributes.

First Party Isolation: Before the introduction of Origin Attributes, the Tor browser had to implement and maintain code for providing First Party Isolation by default within the Tor browser. Tor managed this by maintaining and rebasing 12 changesets with 4,012 lines of code on top of every new Firefox ESR version. Because of resource constraints and the complexity of Firefox internals, Tor's implementation of First Party Isolation was incomplete and did not properly isolate all the different third party resources by First Party Domain. For example, Tor developers were not able to isolate HTTP Cookies or HTTP Authentication by First Party Domain, and hence had to disable third party cookies and third party HTTP Auth in their browser. Aside from the considerable amount of resources and work to provide a complete implementation of First Party Isolation, experience shows that the result is highly dependent on Firefox implementation details and therefore effectively infeasible to fully implement without close collaboration with Mozilla engineers.

Once the Origin Attributes framework was built, Firefox was able to add the First Party Isolation feature with only 7 changesets and 1,822 lines of code. One can see that the amount of work necessary is considerably less than Tor Browser's implementation. In addition, the feature is more complete and less error prone, since Firefox implementation of Origin Attributes covers all security contexts and hence ensures all potential tracking mechanisms are isolated by first party. By using Firefox's implementation of First Party Isolation, the Tor browser can now also re-enable third party cookies and third party HTTP Auth.

6.2 Additional Consumers of Origin Attributes

In addition to Containers and First Party Isolation, site operators and browser vendors have also attempted to implement other features that segregate the origin beyond the SOP specification in order to improve privacy or security on the web. Instead of providing custom implementations for all of the following specifications, they all can build upon Origin Attributes. An additional benefit to using Origin Attributes is that fixing a corner case for one of those implementations not only fixes the problem for the one implementation, but also for all other implementations leveraging Origin Attributes within Firefox.

Private Browsing Mode: The privacy enhancing feature *Private Browsing Mode* (Mozilla, 2009) (also called Incognito or InPrivate mode) allows the user to browse the web using a fresh new Cookie Jar, in memory only, where the browser does not store Internal Cache data, History, or HTTP Cookie data to disk. This allows end users to browse the web without using the regular Cookie Jar, but a temporary one, which is deleted when browsing activity is completed. This is currently being integrated within Firefox's Origin Attributes with *mPrivateBrowsingId*; a value of 0 denotes that Firefox operates in normal mode, while a value of 1 denotes Firefox operates in Private Browsing Mode. In the future, Firefox intends to allow for multiple Private Browsing sessions to occur simultaneously by expanding the value of this id to other integers greater than 1.

SafeBrowsing: Google's *Safe Browsing* (Google, 2012) service enables applications to check URLs against Google's constantly updating lists of suspected phishing, malware, and unwanted software pages. The browser checks every request against this set of lists and then safeguards users by warning them before they click links to dangerous pages and downloads.

The Safe Browsing mechanism within Firefox updates its blacklists in the background every 30 minutes.

To maintain users privacy, Firefox does not want to send identifiable user data to `https://google.com` when contacting the Safe Browsing service, which is also hosted on `https://google.com`. Hence, Firefox extends the `https://google.com` origin with a custom, unique `mAddonId` Origin Attribute reserved for the sole purpose of segregating the Safe Browsing cookie in the browser's Cookie Jar so that a users identifiable cookies from `https://google.com` applications are not sent when updating the Safe Browsing lists of malicious sites.

SubOrigins: Recently, the World Wide Web Consortium (W3C) started discussions about standardizing the concept of *Suborigins* (Weinberger and Akhawe, 2016). Suborigins define a mechanism for programmatically defining namespaces to isolate different applications running in the same physical origin. User agents can extend the SOP with this new namespace to create a security boundary between resources that have the same scheme, host, and port, but different namespaces.

Isolate-Me: More recently, the W3C also started a discussion around an *Isolate-Me* (Stark et al., 2016) specification. This specification proposes a mechanism that gives sites the ability to isolate themselves from other web content. This mechanism provides better security for critical web applications that are willing to trade some of the features of being on the open web, e.g. full linkability, for better cross-site-scripting (XSS) and cross-site request forgery (CSRF) protection.

7 RELATED WORK

User Tracking/Personalization: Several studies have identified the problematic situation of online tracking (Lerner et al., 2016; Libert, 2015; Englehardt et al., 2015; Acar et al., 2014) and all of their results draw a similar picture, concluding that web tracking is omnipresent whenever we surf the web. Englehardt et al. (Englehardt and Narayanan, 2016) provide the largest and most detailed analysis of online web tracking to date. In their study they measure stateful (cookie-based) tracking and show that the level of tracking varies considerably between sites, but conclude that web tracking occurs on almost every site.

Lecuyer et al. (Lécuyer et al., 2014) promises to provide some transparency for end users by allowing users to pinpoint which data is used for online tracking. They developed a system called *XRay*, which predicts which data is being used to generate targeted outputs, such as ads. Similarly, Data et al. present a system called *AdFisher* (Datta et al., 2015), an automated tool that explores and highlights the interaction of user behavior and targeted ads. Both of the systems, *XRay* and *AdFisher*, do bring some light into the dark by providing the user with information on how their data is used and collected for personalization purposes.

A novel approach to tackle the problem of online tracking is taken by Yu et al. (Yu et al., 2016). Their approach departs from the traditional backlist approach which tries to identify third party content, and instead collectively determines if data elements present in a request are safe to submit. Hence stopping web tracking all together.

Extending the Web Security Model: Another feature that could benefit from our presented Origin Attributes is the specification of COWL (Confinement with Origin Web Labels (Stefan et al., 2014)). COWL allows sites to specify privacy and security policies in the form of labels which instruct the browser to enforce additional origin separation within mashups.

Xu et al. (Xu et al., 2015) as well as Zhao et al. (Zhao and Liu, 2015) revealed that Private Browsing Mode is not as private as it claims, and fails to provide adequate or even the intended privacy protection for end users. Both works highlight current implementation shortcomings of Private Browsing Mode which further motivates our work.

Further work that motivates Origin Attributes was presented by Jackson and Barth (Jackson and Barth, 2008) back in 2008. They have shown that the SOP is missing security guarantees for documents on the same origin when those documents have different security characteristics.

8 CONCLUSION AND OUTLOOK

We have implemented Origin Attributes within Firefox, which allows new features and specifications to extend an origin without requiring browser vendors to write and maintain custom patches for every single one of these features. By anchoring Origin Attributes deeply within a browsers rendering engine, we reduce the likelihood of introducing new privacy vulnerabilities with each new implementation of a specification.

To confirm our assumptions that implementing origin extending features on top of the presented Origin Attributes requires only minimal engineering effort, we have presented implementations of Containers and First Party Isolation. Containers allow users to shape their online identity by separating their browsing contexts into isolated Containers. First Party Isolation isolates browsing contexts by the top-level domain the user visits to prevent embedded content from tracking users across sites. We have shown how both of these features have benefited from Origin Attributes, making them more easily maintainable and less error prone.

Finally, we have surveyed newly developing features that could also leverage Origin Attributes to simplify their implementations in the future.

ACKNOWLEDGEMENTS

Thanks to everyone in Security Engineering at Mozilla for their feedback, reviews, and provoking discussions. In particular, thanks to Jonas Sicking, Boris Zbarsky, Olli Pettay, Dan Veditz, Paul Theriault, Ethan Tseng, Steven Englehardt, Monica Chew, Richard Barnes, and Eric Rescorla. Finally, also thank you to the Engineers Jonathan Hao, Tim Huang, Yoshi Huang, Dave Huseby, and Ehsan Akhgari.

REFERENCES

Acar, G., Eubank, C., , Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. (2014). The Web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the Conference on Computer and Communications Security*. ACM.

Barth, A. (2011). HTTP State Management Mechanism. https://tools.ietf.org/html/rfc6265. (checked: October, 2016).

Barth, A., Jackson, C., and Hickson, I. (2009). The Web Origin Concept. https://tools.ietf.org/html/draft-abarth-origin-06. (checked: October, 2016).

Datta, A., Carl, M., Tschantz, C., and Datta, A. (2015). Automated Experiments on Ad Privacy Settings - A Tale of Opacity, Choice, and Discrimination. In *Proceedings on Privacy Enhancing Technologies*. USENIX Association.

Englehardt, S. and Narayanan, A. (2016). Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the Conference on Computer and Communications Security*. ACM.

Englehardt, S., Reisman, D., Eubank, C., Zimmerman, P., Mayer, J., Narayanan, A., and Felten, E. W. (2015). Cookies That Give You Away: The Surveillance Implications of Web Tracking. In *World Wide Web Conference*. ACM.

Google (2012). Safe Browsing. https://developers.google.com/safe-browsing/. (checked: October, 2016).

Jackson, C. and Barth, A. (2008). Beware of finer-grained origins. http://www.adambarth.com/papers/2008/jackson-barth-b.pdf. (checked: October, 2016).

Lécuyer, M., Ducoffe, G., Lan, F., Papancea, A., Petsios, T., Spahn, R., Chaintreau, A., and Geambasu, R. (2014). XRay: Enhancing the Web's Transparency with Differential Correlation. In *Proceedings of the USENIX Security Symposium*. USENIX Association.

Lerner, A., Simpson, A. K., Kohno, T., and Roesner, F. (2016). Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *Proceedings of the USENIX Security Symposium*. USENIX Association.

Libert, T. (2015). Exposing the Hidden Web: An Analysis of Third-Party HTTP Requests on One Million Websites. *International Journal of Communication*.

Mozilla (2009). Private Browsing. https://support.mozilla.org/en-US/kb/private-browsing-use-firefox-without-history. (checked: October, 2016).

Mozilla (2016). Firefox Containers. https://wiki.mozilla.org/Security/Contextual_Identity_Project/Containers. (checked: October, 2016).

Perry, M., Clark, E., and Murdoch, S. (2016). Cross-Origin Identifier Unlinkability. https://www.torproject.org/projects/torbrowser/design/#identifier-linkability. (checked: October, 2016).

Stark, E., West, M., and Weinberger, J. (2016). Isolate-Me. https://wicg.github.io/isolation/explainer.html. (checked: October, 2016).

Stefan, D., Yang, E. Z., Marchenko, P., Russo, A., Herman, D., Karp, B., and Mazieres, D. (2014). Protecting users by confining JavaScript with COWL. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*.

The Tor Project (2012). Tor (anonymity network). https://www.torproject.org/. (checked: October, 2016).

Tran, M., Dong, X., Liang, Z., and Jiang, X. (2012). Tracking the Trackers: Fast and Scalable Dynamic Analysis of Web Content for Privacy Violations. In *Applied Cryptography and Network Security*. Springer.

Weinberger, J. and Akhawe, D. (2016). Suborigins. https://w3c.github.io/webappsec-suborigins/. (checked: October, 2016).

Xu, M., Jang, Y., Xing, X., Kim, T., and Lee, W. (2015). UCognito: Private Browsing Without Tears. In *Proceedings of the Conference on Computer and Communications Security*. ACM.

Yu, Z., Macbeth, S., Modi, K., and Pujol, J. M. (2016). Tracking the Trackers. In *International Conference on World Wide Web*.

Zhao, B. and Liu, P. (2015). Private Browsing Mode Not Really That Private: Dealing with Privacy Breach Caused by Browser Extensions. In *International Conference on Dependable Systems and Networks*. IEEE.