Sandboxing Trackers with Tracking Protection

Steven Englehardt

1. Abstract

We want to provide a tracking protection feature that has *less breakage* and preserves at least some of the revenue stream advertisements provide a publisher. We don't want to block resources that track, we only want to block their ability to track. Our evaluation focuses purely on technical solutions.

We find that sandboxing tracking resources is either ineffective or infeasible depending on the configuration. Sandbox configurations which don't provide Javascript sandboxing either continue to block a large percentage of resources or fail to significantly impact the level of tracking. Sandboxing javascript has the potential to both reduce the level of tracking and amount of breakage, but requires a design that's heavily coupled to current implementation of the tracking scripts.

2. Background

Firefox's Tracking Protection feature [1] currently blocks a resource if its hostname (or any sub-component of the hostname) appears on the tracking protection list and the resource is considered third-party. Tracking protection does not make blocking decisions based on the content type of the resource, but we explore these options in our analysis below.

Tracking Protection relies on Disconnect to curate a list of trackers. Disconnect's definition of a tracker is given in [2], alongside the list of trackers that are blocked and not blocked. A tracker is able to avoid blocking by respecting the EFF's DNT standard [3], which requires the tracker to publicly commit to the statements detailed in the EFF's standard.

A resource is able to track a user using any of the following mechanisms:

- access to storage (stateful tracking)
- javascript access (active stateless tracking)
- access to connection information such as HTTP headers or IP address (passive stateless tracking).

In this study, we address the first two (stateful and active stateless tracking) and leave changes to passive stateless tracking to future work. Passive content, such as image loads, can only tracking using HTTP cookies. Thus, to protect against tracking it's sufficient to strip cookies from the image rather than block it outright. Active content, such as javascript, has many more opportunities to identify a user. We explore different approaches to reducing the ability of active content from tracking while still allowing it to load.

The term *breakage* is often used to describe a first-party site failing to load or work correctly after blocking advertisements and other third-party tracking content. For the purposes of this analysis we expand this definition to include any visible portion of the site failing to load, including advertisements and third-party widgets. This better reflects the fact that the publisher intends for their site to have advertisements and thus blocking them is an instance of breakage.

3. Sandboxing Options Explored

We explored several options to prevent content that appears on the tracking protection list from tracking. The proposed designs vary from allowing and attempting to sandbox all resource loads to only allowing certain content types in certain contexts.

1. Allow all requests; strip cookies
2. Allow all requests; strip cookies and sandbox tracking iframes
3. Block active content embedded in the first-party context; sandbox tracking iframes
4. Sandbox active content embedded in the first-party context; sandbox tracking iframes

To implement these four configurations we rely on several technical solutions detailed in Section 4. Our analysis of these configurations is given in Section 5.

4. Design and implementation details

Stripping and blocking cookies. Cookies can be stripped by adding the LOAD_ANONYMOUS flag to a channel. Although we use this in our prototype, it should be noted that the LOAD_ANONYMOUS flag actually isn't perfect as is. As Wladamir Palant points out [4], it will break proxies which require authentication. For a non-prototype implementation a separate, less restrictive flag would be needed.

Access to storage by third parties is currently controlled by the third-party cookie blocking policy. For example, when set to not allow third-party cookies, javascript in a third-party iframe is not able to get/set cookies with `document.cookie` and does not have access to localStorage. Although not implemented in our prototype, this could be used to block third-party storage access to Javascript loaded in a tracking iframe.

In our proposed design, any channel that is classified as tracking will do an internal redirect where the LOAD_ANONYMOUS flag is added and the LOAD_CLASSIFY_URI flag is removed to prevent re-classification. Stripping cookies is often sufficient to prevent tracking by resources embedded only as passive content.

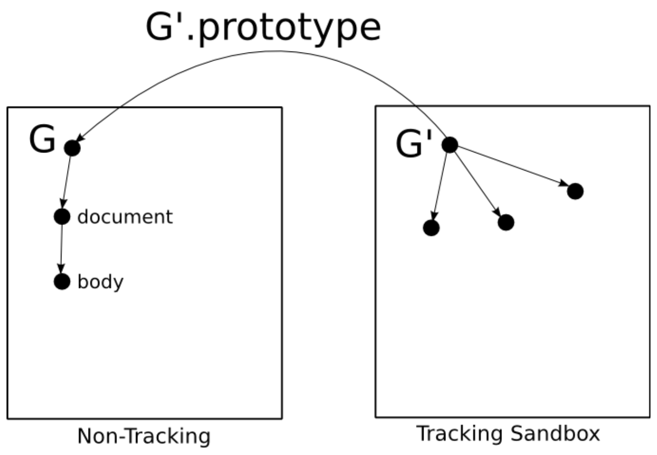
Sandboxing iframes. It is currently possible to apply a set of sandbox flags to iframes [5]. The list of currently supported flags is given in [6] and includes the ability to restrict content in an iframe from running scripts, using modal dialogs, or entering fullscreen. The most relevant restriction is SANDBOXED_ORIGIN, which gives the iframe a null principal. When set, the iframed content will fail same origin checks with any other frame and will not have access to any of the normal storage locations.

In our proposed design, a channel which is classified as tracking will have a SEC_TRACKING_SANDBOXED flag added to its loadInfo security flags. When set on iframe resource requests (and any other request of content type SUB_DOCUMENT), the flag inherits to all subsequent loads within the document. If this flag is present during Document construction, tracking sandbox flags are merged with any normal sandbox flags inherited from the Document's DocShell.

We implement a prototype which sandboxes tracking iframes by applying SANDBOXED_ORIGIN and SANDBOXED_MODALS to any iframe loaded in a channel with SEC_TRACKING_SANDBOXED set. Although fine for testing purposes, the following changes would be necessary for a deployed solution:

- Storage access should not raise a SecurityError, but rather get and set from ephemeral storage
- More of the current sandboxed flags should be considered, such as SANDBOXED_PLUGINS
- New sandbox flags should be added to reduce apply some active fingerprinting restrictions, perhaps making use of the uplifted Tor patches [7].

Sandboxing Javascript. Gecko provides the machinery to sandbox scripts [8], namely the ability to create non-DOM globals, to define wrappers for interactions across globals, and a principal hierarchy that governs which wrappers apply between two given globals. To sandbox tracking scripts, we can define a separate global from the main page and give filtered access to methods, objects, or DOM defined in the main context through custom wrappers. Example wrappers may deny-all (opaque wrappers), the ability to see through expando properties (XrayWrappers), or only provide access to certain APIs (filtering wrappers). See [9] for more information on principal types and their security relationships.



In our proposed design, when a tracking script is loaded into the first-party context (i.e. not in an iframe) we intercept the load and place it in a *tracking sandbox*. The tracking sandbox is a separate compartment from the *non-tracking context*, and has it's own global who's prototype (effectively) points to the global of the non-tracking context. Although it's conceptually useful to think of the prototype of G' pointing to the DOM's Window, there are implementation differences. XPConnect does not allow inherited methods in the tracking sandbox to work with the non-DOM derived |this| of G'. Instead, a custom proxy is used to remap |this| and other objects. See [10-12] for more information.

Any methods defined by the tracking script live in the sandboxed context and are unavailable to the main page. Access in the reverse direction (e.g. when a tracking script attempts to interact with the DOM) will be cross-sandbox, and is thus subject to any *tracking wrappers* we define. We explore possible wrappers and sandbox configurations in Section 5.

5. Sandbox Configurations

(1) Allowing all requests while stripping / blocking cookies

This is similar to third-party cookie blocking, except that only third-party cookies from known trackers would be blocked. This has the potential for less breakage as first-party resources on separate domains, e.g. `example-img.com` on `example.com` would still load cookies, while third-party trackers would no longer have access to cookies.

Active content in the first-party context can track users by utilizing first-party DOM and storage access. When third-party state is blocked, scripts (and other active content) can use other tracking techniques. Those embedded in a third-party iframe will no longer have access to third-party state such as cookies or localStorage, however third-party scripts embedded into the first-party context will have full access to the first-part DOM and storage. This opens up several attacks to identify the user across sites: the script could scan the DOM for identifiers such as username or email addresses or could rewrite all cross-domain links to include an identifier their script could check for on the subsequent page load.

Active content can fingerprint users. Scripts embedded in the first-party context could associate this fingerprint with a first-party ID cookie and would only need to re-fingerprint on subsequent cookie clears.

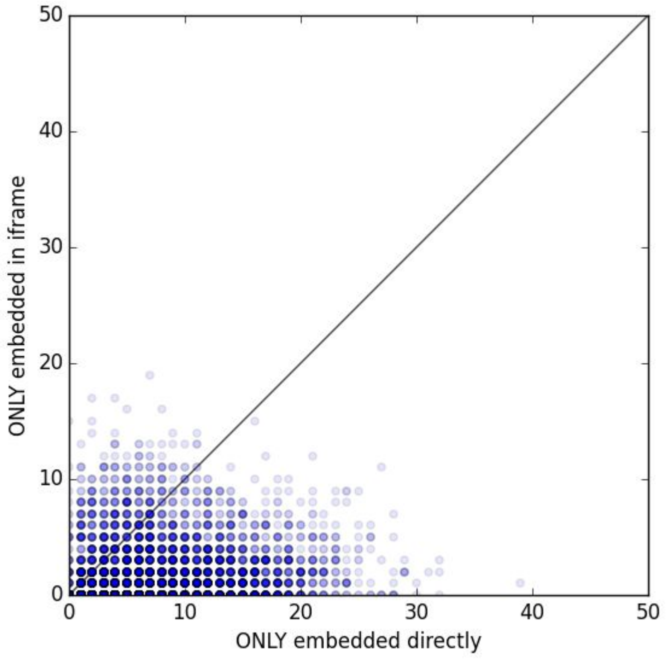
Enabling this feature by default will encourage fingerprinting. Users have control over cookies and other browser state, but can't control fingerprinting. See: *If you're going to track me, please use cookies* by Ed Felten [13]. A hybrid blocking solution may be possible, where trackers are allowed to load with storage blocked and any trackers found fingerprinting will have their resourced blocked by default. However this solution will either require reliance on some external party to detect fingerprinting (as I do in my research [14]) or will require Mozilla to build fingerprinting detection into the browser.

(2) Allow all requests; strip cookies and sandbox tracking iframes

This provides additional benefits above (1) in that any script loaded into a sandboxed iframe will not have the ability to do stateful or active stateless tracking. As discussed in (1), scripts loaded into the first-party context will still have the ability to track users both by using first-party identifiers or by fingerprinting. The effectiveness of this solution depends on the number of parties (and the number of instances of a single party) embedded directly versus embedded in a sandboxed frame.

A low number of trackers experience a meaningful drop in their tracking ability. This feature is only effective if a small number of well-regulated third parties are embedded into the first-party and the remainder are within a tracking iframe. We show that this isn't the case by crawling the top 10,000 sites, of which we analyze the homepages of the top 8,683. On these sites there are 1,073 distinct third-party PS+1s which embed tracking scripts, 640 of which embed scripts on at least two sites in our sample. Tracking scripts are embedded far less frequently in an iframe than they are directly into the first-party context.

Approximately 25% of tracking scripts embed iframes which are same origin to their script. The figure below plots first-party sites by the number of trackers they embed directly versus within an iframe (after removing trackers embedded in both contexts). There is a clear skew towards direct embedding, in fact only 6% of first-parties sampled have more scripts embedded in tracking iframes than not.



The table below shows the percentage reduction in the number of sites a third-party would be able to track on if this feature were enabled.

Reduction of [0,10)% of sites: 452 (71%) third parties
Reduction of [10-20)% of sites: 50 (8%) third parties
Reduction of [20-30)% of sites: 21 (3%) third parties
Reduction of [30-40)% of sites: 37 (6%) third parties
Reduction of [40-50)% of sites: 21 (3%) third parties
Reduction of [50-60)% of sites: 21 (3%) third parties
Reduction of [60-70)% of sites: 10 (2%) third parties
Reduction of [70-80)% of sites: 7 (1%) third parties
Reduction of [80-90)% of sites: 6 (1%) third parties
Reduction of [90-100)% of sites: 5 (1%) third parties
Reduction of 100% of sites: 10 (2%) third parties

The table above shows that most trackers (71%) have a very low reduction (10% or less) in the amount of sites they are able to track on. A relatively small number of trackers (10%) have their ability to track reduced by 50% or more. While this configuration provides additional tracking protection above (1), we do not feel the reduction in tracking justifies the engineering work required nor the potential breakage that will occur within iframes.

Strong fingerprinting restrictions on iframes may cause third-party services to lobby for direct embedding. For example, the impression verification company DoubleVerify is loaded by DoubleClick ads primarily within an iframe (369 vs 121 in our dataset) and uses fingerprinting. If this configuration were to be enabled by default, they could lobby DoubleClick to change their embedding practices. Even ignoring cross-party collaboration, ~25% of tracking scripts are embedded into an iframe which was itself embedded by a script from the same tracker.

(3) Block active content in the first-party context; sandbox tracking iframes

Blocking active content in the first-party context effectively prevents third-party scripts from being able to track users using stateless methods. The only tracking scripts which are allowed to load will be contained in a tracking iframe which has restricted access to APIs known to be used for fingerprinting. These scripts will also not have access to the main page's DOM, limiting their ability to track users through first-party information.

The level of breakage is prohibitively high. Many of the major third-party services request that you embed them as a script. For example Google [15], Facebook [16], and Twitter [17]. Some of these parties also provide direct iframe options that don't require script access, and we found a few sites using these: wetter.com and redmondpie.com are two examples for Doubleclick and Facebook. On these sites the advertisements and like buttons will load under this configuration. However these are very small in number: just 2 / 10,000 for DoubleClick and 209 / 10,000 for Facebook.

In fact, the data we collected shows that at least 45% of sites contain tracking iframes which are embedded by tracking scripts. This should be considered a very conservative lower bound as doubleclick iframes are embedded in a way that we can't currently track and are not included in this estimate.

We have implemented a prototype of this functionality, available in Bug 1298207. Anecdotally, most major pages appear to have the same level of breakage when browsing with this patch applied as they do with standard tracking protection.

(4) Sandbox active tracking content in the first-party context; sandbox tracking iframes

Sandboxing active tracking content embedded in the first-party context could solve the breakage problems experienced in (3) while avoiding the privacy holes of (2) and (1). However we were not able to derive a feasible design that would have sufficiently low breakage, prevent trivial sandbox escaping, and have a reasonable engineering and maintenance cost.

Access to the main page by tracking scripts is restricted by the filtering wrappers applied across the sandbox membrane when going from the tracking sandbox to the non-tracking context. Due to the complexity of DOM interaction, we default to deny-all and whitelist specific actions. The audit process requires collecting all calls that require DOM interaction from all tracking scripts. Each new call added to the whitelist would need to be analyzed to see if it can be combined with other calls on the whitelist to escape the sandbox.

We did not do a full audit of scripts and instead analyze a few specific cases as a feasibility check. We chose to analyze several Google script interactions given their popularity as an advertisement provider.

Popular scripts require inline javascript to properly load. One of Google's advertising scripts (http://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js) inserts an iframe without a src attribute. This script is present on nearly 10% of the top sites. Instead the iframe is navigated using the the following inline function set in the `onload` handler:

```
var i = this.id,
      s = window.google_iframe_oncopy,
      h = s & amp; & amp;
s.handlers, h = h & amp; & amp;
h[i], w = this.contentwindow, d;
try {
      d = w.document
} catch (e) {}
if (h & amp; & amp; d & amp; & amp;
      (!d.body || !d.body.firstchild)) {
      if (h.call) {
      settimeout(h, 0)
      } else if (h.match) {
      try {
            h = s.upd(h, i)
      } catch (e) {}
      w.location.replace(h)
      }
}
```

Note that this function requires access to `window.google_iframe_oncopy` and subsequently uses the result to navigate itself. With our suggested sandbox construction, we would need to allow tracking scripts to inject elements into the DOM with arbitrary inline javascript and would require tracking functions to be accessible to those inline scripts.

However, whitelisting inline functions would provide a way for scripts to escape the tracking sandbox. Instead, we could rewrite any inline function and define it in the tracking context. This will force it to interact with the DOM through our filtering wrappers. Note that we also need to support any calls made by the inline function across the process boundary. e.g. for the example above we need to support editing the `location` attribute of any element in the DOM. In order for inline function re-writing to be supported we would need some machinery to reflect functions defined in the tracking sandbox into the main page; see the discussion below for more details.

Similarly, a second advertising script from Google (http://partner.googleadservices.com/gpt/pubads_impl_91.js), which is present on ~25% of the top first party sites, injects iframes with the following `javascript:` src attribute.

```
src="javascript:&quot;<html><body
style='background:transparent'></body></html>&quot;"
```

This script also injects iframes with large scripts serialized into the name attribute. These scripts are presumably passed to eval at a later point, but we did not trace the execution any further.

Additionally, many scripts which are sandboxed inject inline javascript. In total, around 18% of homepages have tracking javascript which injects inline scripts. All of these scripts would also require inline function rewriting and subsequent reflection.

Popular tracking scripts load other scripts by inserting <script src=...> tags into the DOM. This can be used to dynamically load a tracking script from an origin which is not currently on the tracking protection list, making the sandbox trivially avoidable by an adversary which has a large number of hosts at their disposal.. Disabling the ability to inject script elements with the src set would cause a large amount of breakage. Our measurements show that around 63% of the top 10,000 sites include tracking scripts which inject other scripts into the DOM.

Functions defined in tracking scripts need to be reflected into the main context. Machinery currently exists to do this for certain addons in e10s windows [18]. However, this is not exposed to the web and my understanding is that there is no desire to expose it to the web [19]. If functions defined in tracking scripts were reflected into the main context using this machinery, we would be able to rewrite inline javascript from inserted elements to functions defined in the tracking context as discussed above. When a rewritten function is called in the main context, it would call the reflected version which would execute in the tracking context.

5. Conclusion

The analysis presented above shows that all possible sandboxing solutions will either provide insufficient tracking protection to users (1) and (2), will continue to break too much of the web (3) and (4), or will use a complex and brittle architecture which requires constant maintenance (4). Thus, the current embedding practices of third-parties is incompatible with the current sandbox approaches.

Mozilla could consider non-technical solutions, such as publishing a criteria that ads must follow to work under sandboxing. In the absence of a purely technical solution, the current Tracking Protection model of blocking all resources and whitelisting third-parties which support DNT seems preferable. It is much easier to understand, doesn’t mix technical and non-technical solutions, and has been designed to aid in legal enforcement against bad actors.

6. Prototype and Data Availability

Bug 1298207 [20] includes the prototype patch and links to the data collection and analysis framework

References

[1] https://wiki.mozilla.org/Security/Tracking_protection
[2] https://disconnect.me/trackerprotection
[3] https://www.eff.org/dnt-policy
[4] https://adblockplus.org/blog/why-you-do-not-want-to-use-the-load_anonymous-flag
[5] https://html.spec.whatwg.org/multipage/embedded-content.html#attr-iframe-sandbox

[6] https://dxr.mozilla.org/mozilla-central/source/dom/base/nsSandboxFlags.h
[7] https://wiki.mozilla.org/Security/Tor_Uplift/Tracking
[8] https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Language_Bindings/Components.utils.Sandbox
[9] https://developer.mozilla.org/en-US/docs/Mozilla/Gecko/Script_security
[10] http://searchfox.org/mozilla-central/rev/d83f1528dbcb1e837d99cf5b6c36a90b03bf0e77/js/xpconnect/src/Sandbox.cpp#649
[11] http://searchfox.org/mozilla-central/rev/d83f1528dbcb1e837d99cf5b6c36a90b03bf0e77/dom/base/nsGlobalWindow.cpp#8277
[12] https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Language_Bindings/Components.utils.Sandbox#options
[13] https://freedom-to-tinker.com/blog/felten/if-youre-going-track-me-please-use-cookies/
[14] https://webtransparency.cs.princeton.edu/webcensus/#fp-results
[15] https://support.google.com/adsense/answer/181950?hl=en
[16] https://developers.facebook.com/docs/plugins/like-button
[17] https://publish.twitter.com/#
[18] https://hg.mozilla.org/mozilla-central/rev/7b44740b12b1#l6.49
[19] https://hg.mozilla.org/mozilla-central/rev/7b44740b12b1#l8.98
[20] https://bugzilla.mozilla.org/show_bug.cgi?id=1298207