Project: Tracker blocker browser extension

Background & Requirements

Glossary:

- **Request**: when a user loads a page, that page can contact different other websites in the background for different purposes. We call those contact attempts "requests".
- **Legitimate request:** for the purposes of this document, legitimate requests are the ones that don't compromise the user's privacy. For example, a request to load an image, a custom font, a needed script for the page's application, etc.
- **Illegitimate request**: conversely, an illegitimate request is the one that would compromise the user's privacy – a tracker.
- **A user:** a regular, non-tech savvy user navigating the web and hopefully future user of our product. As a regular user they do not need to know about a page's lifecycle at all.
- **User script**: A Javascript file a user can install in their browser, via a user script manager[1], in order to customize or modify appearance or behavior in webpages they visit.

[1] Like GreaseMonkey, TamperMonkey, ViolentMonkey, etc.

- **Browser extension**: Similar to a user script, an extension is able to change appearance or behavior in webpages but it has more powerful and versatile APIs than user scripts. It allows us to tap in different stages of a page's lifecycle that a user script wouldn't.
- **WebExtensions**: Mozilla's API to build cross-browser compatible extensions. It is currently in a W3C draft[2] proposal to become a web standard.

Requirements:

1. **A metrics service**. This service would receive metrics from the extension and the extension's page. The metrics would help us measure both the extension's efficiency (number of trackers blocked, number of false-positives, etc) and user engagement (number of devices/users using our extension, adoption growth over time, etc).
2. **A public storage for the extension**. Later on we can use each browsers' Add-on Stores to distribute the extension, but we need to host it ourselves so they can fetch it and sign it.
3. **Developer accounts** with Mozilla, Google, etc to publish and manage our extension with them.

Constraints:

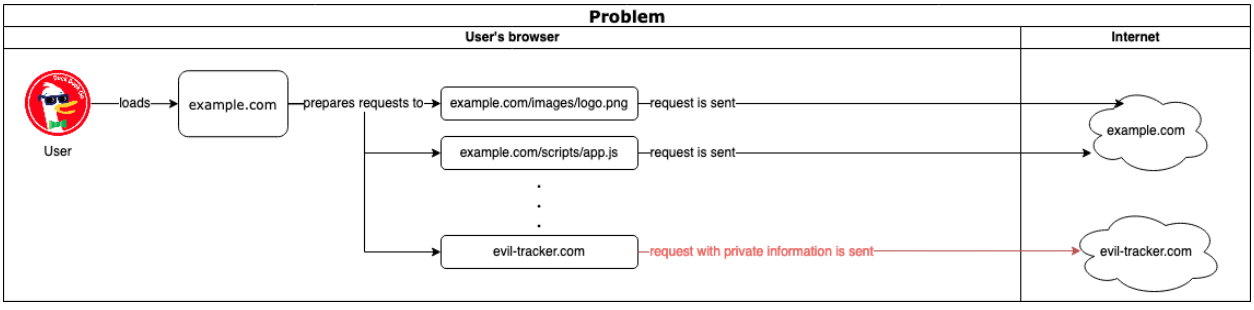
1. The extension must either completely anonymize or block illegitimate requests before they leave the user's browser.
2. The extension should work with at least the last two versions of major browsers (Firefox, Chrome, Edge and Opera).
3. The extension itself must not send any information that might identify the user.

Concurrent work (out of scope of this document):

1. **Engagement via product page.** Before we gauge the users' interest by releasing the extension and then collecting metrics, we could start in parallel a marketing campaign on our website and social media by creating a page for our extension. We could collect metrics like page views, browsers used to visit our page, etc. Those would assist us in our decisions (e.g.: which browser platform to initially focus our efforts on). After releasing the extension, we would be able to augment those metrics with the extension's, making even better-informed decisions for our next steps.

Problem statement:

When visiting a website, a user has no way of knowing what requests will be made on their behalf and which information about themselves will leave their browsers. Users that care for their privacy and want to ensure their right to it have no way of doing so.



[2] https://wiki.mozilla.org/WebExtensions/FAQ#Is_the_WebExtensions_API_a_Web_standard.3F

Enter our extension! With our extension we will be able to discern legitimate from illegitimate requests on a particular website and prevent the illegitimate ones from sharing private information about our users. Hooray!

Approaches:

The final recommended approach is in the section 2.1 below and the scope for this project is highlighted in the "High level use cases and test cases" section below.

1) *[Not considered]* User Scripts

Pros:

1. Simple and faster to develop and release compared to browser extensions
2. Easier to install if a user already has user script manager installed.

Cons:

1. Deal breaker: user scripts don't provide the APIs needed to intercept requests and either modify or cancel them.
2. Additional friction to install if the user doesn't have a user script manager installed.
3. Engagement & branding. We can't draw the user's attention (when we block a request, for example) since the icon and badge in their browser toolbar will be the user script manager's and not ours. This might further make it difficult for our users to interact with our solution and, for example, toggle the tracker protection on or off, submit a report, etc.
4. Performance. Compared to browser extensions, a user script will have a slight overhead on performance due to the user script manager parsing the script and checking for permissions within it before actually executing it. Plus the memory overhead in case a user had to install the script manager only to be able to use our user script.

2) Browser extension

Pros:

1. All "cons" listed under "User Scripts" above reversed.
2. An extension (using WebExtensions) increases our reach since it's more likely that a browser would implement an open standard (especially if it becomes a W3C one) while not every browser has a user script manager available and not all user script managers provide the same APIs.
3. Although still a draft, all major browsers already implement the API we will need for our extension[3]

Cons:

1. More complex and slower to develop and release (compared to user scripts).
2. Since not all browsers fully implement the WebExtensions standards yet, we might need a couple of branches in the code to provide cross-browser compatibility. Fortunately, Mozilla also provides a polyfill that takes care of most of those cases[4].

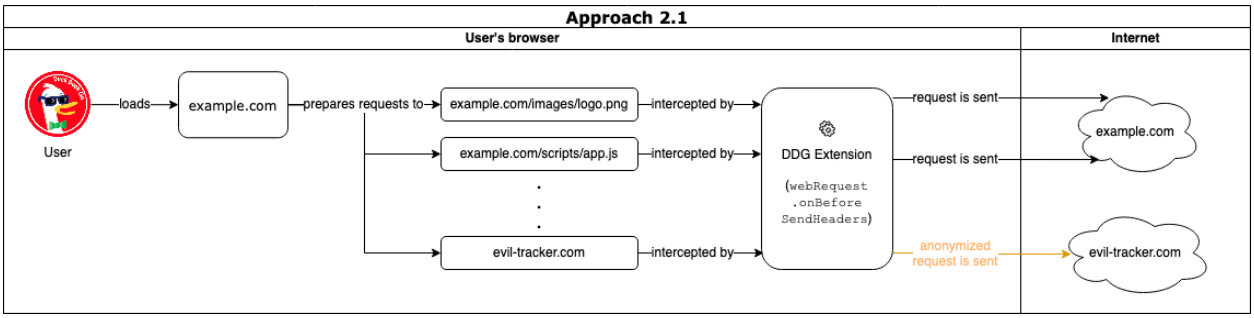
[3] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/onBeforeRequest#Browser_compatibility

[4] https://github.com/mozilla/webextension-polyfill

There are two possible routes we might follow by using an extension:

2.1) *[Not considered]* Anonymizing the request

Instead of completely blocking an illegitimate request we could consider anonymizing it instead: we would still send the request, but modifying the request's header, stripping away all private data. Ideally we would provide both options and let the user decide how strict they want to be about their privacy.



Pros:

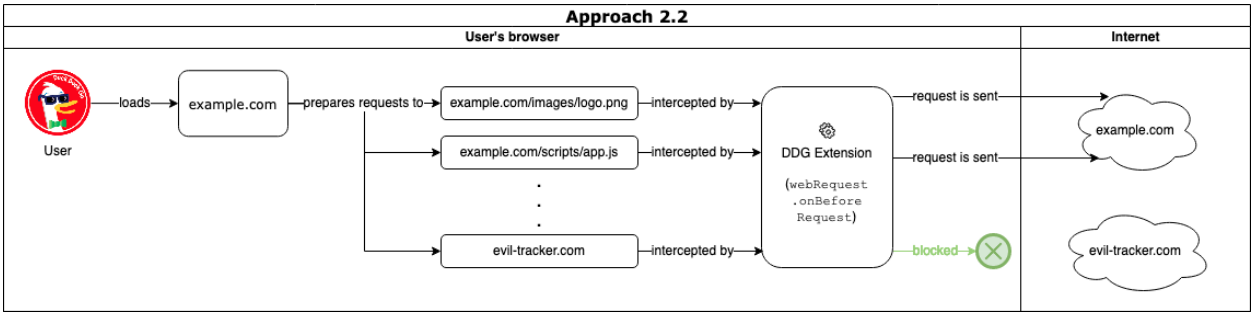
1. Sometimes tracker scripts actually provide useful or even necessary content to a page. By blocking them, we risk breaking the page to a point of becoming unusable. By anonymizing the request and allowing it to go through, we would mitigate that risk.

Cons:

1. Deal breaker: only one browser extension can alter a request header at a time[5]. So if the user has another extension installed that also modifies the header, there's no guarantee ours will run. This would risk illegitimate requests being sent and hence allowing our user getting tracked against their will, defeating the extension's purpose and breaking the user's trust.
2. An anonymized request to an ill-intentioned tracker can still be "identifiable enough" for tracker companies. Consider the following scenario: a webpage embeds a third-party script that loads Facebook albums. A user visits a webpage that loads a Facebook album about a political party event via that script. Although we might have anonymized the request, Facebook would still know that "someone" in that area (via the user's IP address) is interested in that particular political party and increase the number of ads and news about such political party in that region. Worse yet: other ill-intentioned organizations might purposefully buy those ads to direct fake news to that region.
3. A tracker could be hiding the private information on customized or encrypted headers, making it easy to bypass our anonymization filters.

[5] https://developer.chrome.com/extensions/webRequest#implementation

2.2) *[Recommended]* Blocking the request



Pros:

1. A stricter and safer default (compared to 2.1 above). In the worst-case scenario of breaking a page, the user can still consciously make the decision of disabling the extension for that webpage.

Cons:

- Might break or render a page unusable

Testing:

High level use cases and test cases

<u>Note</u>: I'm merging use cases and test cases below for time's sake. In a real-world scenario, they would be separate ones as they don't completely map one to one.

#	Use / Test case
Essential functionality	
1	Requests matching against the deny list are blocked
2	Requests matching against the allow list pass
3	Requests not matching either of the lists pass
4	Requests matching against both lists take the allow list as precedence and are not blocked
5	Metric is emitted upon blocking a request
6	User can provide feedback via extension (bug reports, feature requests…)
Extension's lifecycle	
7	Installing the extension asks for required permissions
8	Basic functionality works the same across browser restarts
9	Basic functionality works the same across different profiles
10	Test extension automatic update mechanism works
Interactive functionality	
11	User can add entries to the allow list
12	User can add entries to deny list
13	Upon adding entries to allow or deny list, offer to reload current tab, all tabs, or none
14	Metric is emitted upon adding a new entry to the allow or deny lists
15	Custom additions to the lists are persisted across browser restarts (for the profile only)
16	User can disable/enable the extension for the current tab
17	Metric is emitted upon disabling/enabling the extension for a tab
Advanced functionality	
18	The extension is able to update the local deny list from DDG's servers
19	The extension is able to sync custom entries to the lists to the user's account (Google's, Mozilla's, etc) so they persit across devices/browser installations.

20	Ensure compatibility with Google Chrome (desktop)
21	Ensure compatibility with Microsoft Edge (desktop)
22	Ensure compatibility with Apple Safari (desktop)
23	Ensure compatibility with Google Chrome (mobile)
24	Ensure compatibility with Microsoft Edge (mobile)
25	Ensure compatibility with Apple Safari (mobile)
Reporting functionality	
26	Extension can show a simple report with "top hitters" and "total requests blocked over time" – local
27	Extension can show a simple report with "top hitters" and "total requests blocked over time" – worldwide

Table 1 – Use/test cases

Testing tools

- **web-ext**[6] and **Permissions Manager**[7]: to speed up development and reduce human errors when testing:
    - Automates manual steps like installing the development version of the extension on the browser, guaranteeing we're testing the latest changes.
    - Able to run the extension each time in a new profile, avoiding accidental manual configuration persisting in-between tests.
    - Guarantees we're testing the latest changes to the extension's code.
    - Quickly clear or change our extension's permissions in order to test them.
- **Automated unit tests**: to ensure basic functionality works (e.g.: allow/deny lists pattern matching, helper libraries we might write, etc).
- **Automated end-to-end tests**: to ensure the extension work as expected while actually running in a browser. We can use browser drivers like Playwright, Puppeteer, etc.

Scope

For the purposes of this project (DDG's interview process) and given it's my first time developing browser extensions, I believe I can deliver items #1-9 from the table above in the time I have. I will also only focus on a Firefox version of the extension.

The use cases in the table above were prioritized in a way to provide small incremental deliveries to our users (we will have a simple working version right after the first deliverable). Each relevant set of deliverables are accompanied by metrics so we can evaluate their success and adapt our direction accordingly.

Every set of deliverables will be accompanied by as much automated tests as possible. It's paramount we minimize our defects and regression bugs since our mission is around privacy and we should avoid breaking the users' trust as much a possible by offering highly tested, trustworthy releases.

This way, no matter how much time we're able to invest in this project, we'll have a working, value-adding, tested extension we can ship to users.

[6] https://github.com/mozilla/web-ext
[7] https://github.com/rpl/dev-webext-permissions-manager

Additional considerations:

Metrics service scalability:

We need to ensure our metrics service will be able to support the new traffic coming from all the users using the extension and sending their metrics. We can elastically scale up or down our fleet by setting up auto-scaling rules based on the current traffic received. This is out of scope for this document.

Thinking bigger:

WebExtensions APIs provide a few other APIs that are not about blocking trackers but would greatly add value to our privacy extension. We could explore them for future versions of our extension:

- With the `bookmarks` API[8], we could traverse our users' bookmarks and show them a report with each website's score when it comes to privacy / tracking for their awareness.
- With the `privacy` API[9], we could suggest different browser privacy configurations the user might be interested in tuning up to make their navigation even more private.
- With the `dns` API[10], we could add an extra layer of security to our extension, where we could not only deny requests based on hostnames but also the IP addresses they resolve to, so if a tracker managed to not be on our blacklist because they changed their hostname, we could still block them via their known IP address.

[8] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/bookmarks
[9] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/privacy
[10] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/dns