Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies

Gertjan Franken, Tom Van Goethem, and Wouter Joosen, ***imec-DistriNet, KU Leuven***

https://www.usenix.org/conference/usenixsecurity18/presentation/franken

This paper is included in the Proceedings of the 27th USENIX Security Symposium.

August 15–17, 2018 • Baltimore, MD, USA

ISBN 978-1-939133-04-5

Open access to the Proceedings of the 27th USENIX Security Symposium is sponsored by USENIX.

Who Left Open the Cookie Jar?
A Comprehensive Evaluation of Third-Party Cookie Policies

Gertjan Franken
imec-DistriNet, KU Leuven

Tom Van Goethem
imec-DistriNet, KU Leuven

Wouter Joosen
imec-DistriNet, KU Leuven

Abstract

Nowadays, cookies are the most prominent mechanism to identify and authenticate users on the Internet. Although protected by the Same Origin Policy, popular browsers include cookies in all requests, even when these are cross-site. Unfortunately, these third-party cookies enable both cross-site attacks and third-party tracking. As a response to these nefarious consequences, various countermeasures have been developed in the form of browser extensions or even protection mechanisms that are built directly into the browser.

In this paper, we evaluate the effectiveness of these defense mechanisms by leveraging a framework that automatically evaluates the enforcement of the policies imposed to third-party requests. By applying our framework, which generates a comprehensive set of test cases covering various web mechanisms, we identify several flaws in the policy implementations of the 7 browsers and 46 browser extensions that were evaluated. We find that even built-in protection mechanisms can be circumvented by multiple novel techniques we discover. Based on these results, we argue that our proposed framework is a much-needed tool to detect bypasses and evaluate solutions to the exposed leaks. Finally, we analyze the origin of the identified bypass techniques, and find that these are due to a variety of implementation, configuration and design flaws.

1 Introduction

Since its emergence, the Web has been continuously improving to meet the evolving needs of its ever-growing number of users. One of the first and most crucial improvements was the introduction of HTTP cookies [5], which allow web developers to temporarily store information such as website preferences or authentication tokens in the user's browser. After being set, the cookies are attached to every subsequent request to the originating domain, allowing users to remain logged in to a website without having to re-enter their credentials.

Despite their significant merits, the way cookies are implemented in most modern browsers also introduces a variety of attacks and other unwanted behavior. More precisely, because cookies are attached to every request, including third-party requests, it becomes more difficult for websites to validate the authenticity of a request. Consequently, an attacker can trigger requests with a malicious payload from the browser of an unknowing victim. Through so-called cross-site attacks, adversaries can abuse the implicit authentication to perform malicious actions through cross-site request forgery attacks [6, 54], or extract personal and sensitive information through cross-site script inclusion [24] and cross-site timing attacks [9, 16, 48].

Next to cross-site attacks, the inclusion of cookies in third-party requests also allows for users to be tracked across the various websites they visit. Researchers have found that through the inclusion of code snippets that trigger requests to third-party trackers, the browsing habits of users are collected on a massive scale [2,40,53]. These trackers leverage this aggregated information for the purpose of content personalization, e.g. on social networks, displaying targeted advertisements, or simply as an asset that is monetized by selling access to the accumulated data.

As a direct response to the privacy threat imposed by third-party trackers and associated intrusive advertisements, a wide variety of efforts have been made. Most prominently is the emergence of dozens of browser extensions that aim to thwart their users from being tracked online. These extensions make use of a designated browser API [39] to intercept requests and either block them or strip sensitive information such as headers and cookies. Correspondingly, several browsers have recently introduced built-in features that aim to mitigate user tracking. For instance, Firefox in its private browsing mode will by default block third-party requests that

are made to online trackers [7]. It is important to note that the effectiveness of these anti-tracking mechanisms fully relies on the ability to intercept or block *every* type of request, as a single exception would allow trackers to simply bypass the policies. In this paper, we show that in the current state, built-in anti-tracking protection mechanisms as well as virtually every popular browser extension that relies on blocking third-party requests to either prevent user tracking or disable intrusive advertisements, can be bypassed by at least one technique.

Next to tracking protections, we also evaluate a recently introduced and promising feature aimed at defending against cross-site attacks, namely same-site cookies [51]. While cross-site attacks share the same cause as online tracking, i.e. the inclusion of cookies on third-party requests, their defenses are orthogonal. The `SameSite` attribute on cookies can be set by a website developer, and indicates that this cookie should only be included with first-party requests. Consequently, when this policy is applied correctly, same-site cookies defend against the whole class of cross-site attacks. Similar to the tracking defenses, the security guarantees provided by same-site cookies stand or fall by the ability to apply its policies on *every* type of request. As part of our evaluation, we discovered several instances in which the same-site cookie policy was not correctly applied, thus allowing an adversary to send authenticated requests regardless of the `lax` or `strict` mode applied to the same-site cookie. Although this bypass could only be used to trigger GET requests, thereby making the exploitation of CSRF vulnerabilities in websites that follow common best-practices more difficult, it does underline the importance of a systematic evaluation to test whether browser implementations consistently follow the policies proposed in the specification.

In this paper, we present the first extensive evaluation of policies applied to third-party cookies, whether for the purpose of thwarting cross-site attacks or preventing third-party tracking. This evaluation is driven by a framework that generates a wide-range of test cases encompassing all methods that can be used to trigger a third-party request in various constructs. Our framework can be used to launch a wide variety of different browsers, with or without extensions, and analyze, through an intercepting proxy, whether the observed behavior matches the one expected by the browser instance. We applied this framework to perform an analysis of 7 browsers and 46 browser extensions, and found that for virtually every browser and extension the imposed policy can be bypassed. The sources for these bypasses can be traced back to a variety of implementation, configuration and design flaws. Further, our crawl on the Alexa top 10,000 did not identify any use of the discovered bypasses in the wild, indicating that these are novel.

Our main contributions are the following:

- We developed a framework with the intent to automatically detect bypasses of third-party request and cookie policies. This framework is applicable to all modern browsers, even in combination with a browser extension or certain browser settings.
- By applying the framework to 7 browsers, 31 ad blocking and 15 anti-tracking extensions, we found various ways in which countermeasures against cookie leaking can be bypassed.
- We performed a crawl on the Alexa top 10,000, visiting 160,059 web pages, to inspect if any of these bypasses were already being used on the Web. In order to estimate the completeness of our framework, we analyzed the DNS records spawned by each web page.
- Finally, we propose solutions to rectify the implementations of existing policies based on the detected bypasses.

2 Background

A fundamental trait of the modern web is that websites can include content from other domains by simply referring to it. The browser will fetch the referenced third-party content by sending a separate request, as shown in Figure 1. The web page of `first-party.com` contains a reference to an image that is hosted on `third-party.com`. In this scenario, the user first instructs his browser to visit this web page, e.g. by entering the address in the address bar or by clicking on a link. This will initiate a request to the web page `http://first-party.com/`, and a subsequent response will be received by the browser (1). While parsing the web page, the user's browser comes across the reference to `https://third-party.com` and fetches the associated resource by sending a separate request (2). The browser will include a `Cookie` header [5] to the request if these were previously set for that domain (using the `Set-Cookie` header in a response). This applies to both the request to `first-party.com` as well as `third-party.com`. In this scenario, we would name the cookies attached to the latter request *third-party cookies*, as this is a request to a different domain than the including document.

2.1 Cross-site attacks

Because browsers will, by default, attach cookies to any request, including third-party requests, an adversary is

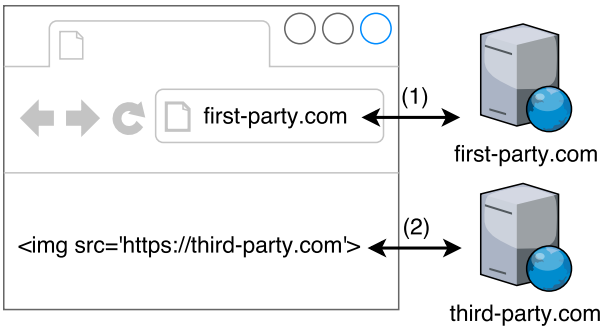

Figure 1: Example of a cross-site request.

able create a web page that constructs malicious payloads which will be sent using the victim's authentication. Through these so-called cross-site attacks, attackers can trigger state changes on vulnerable websites or extract sensitive information.

One of the most well-known cross-site attacks is cross-site request forgery (CSRF). CSRF attacks aim to perform undesirable actions, e.g. transfer funds to the account of the adversary, on behalf of the victim who is authenticated at the vulnerable website. Typically, this will be done by triggering a POST request to the targeted website, as it is considered best-practice to prevent GET requests from having any state-changing effect [15]. Although websites of large organizations such as The New York Times, ING, MetaFilter and YouTube have been found to be vulnerable to CSRF attacks in the past [54], the increased awareness among web developers and countermeasures integrated in popular frameworks resulted in a drastic decrease in vulnerable websites. According to the OWASP Top Ten Project, only 5% of current websites were found to be vulnerable, thus leading to the exclusion of CSRF from the list of the ten most critical web application security risks. Effective countermeasures, such as requiring an unguessable token in requests, have been known for a long period [6, 54], and have been extensively applied [47].

In contrast to CSRF, cross-site script inclusion (XSSI) and cross-site timing attacks aim to derive sensitive information. XSSI attacks bypass the Same-Origin Policy (SOP) in an attempt to obtain information linked to the authenticated user account [24]. Timing attacks, on the other hand, try to construct sensitive data by observing side-channel leaks [9, 16, 48].

A recently proposed mechanism called same-site cookies aims to protect against the whole class of cross-site attacks [51]. Same-site cookies are generic cookies with an additional attribute named `SameSite`. Similar to other cookie attributes, the `SameSite` attribute is determined by the website that sets the cookie. This attribute can be given one of two values: `lax` or `strict`. When the value is set to `lax`, the cookie may only be included in cross-site GET requests that are top-level (i.e. the URL in the address bar changes due to the request). An exception to this is a cross-site request initiated by Prerender functionality [46], in which this cookie is included anyway. When the attribute value is set to `strict`, the cookie may never be included in any cross-site requests.

At the time of writing, same-site cookies are supported by Chrome, Opera, Firefox and Edge [8, 27, 50]. Same-site cookies are backwards compatible; browsers that do not offer support will just treat same-site cookies as regular cookies. This, combined with the fact that same-site cookies are mainly intended as an in-depth defense mechanism, encourages web developers to still employ traditional defenses such as CSRF tokens to thwart cross-site attacks. While the adoption of same-site cookies is still relatively small, with only a few popular websites implementing them [42], the fact that they can mitigate a whole class of attacks makes them a very promising defense mechanism.

2.2 Third-party tracking

Internet users can be tracked for a variety of purposes, often with economic motives as the driving force behind it, e.g. advertising, user experience or data auctioning [26]. One way of employing online tracking is through embedded advertisements, which include tracking scripts to learn more about the user's interests and personalize the advertisements based on this information. Alternatively, website administrators may include scripts from analytic services, which gather insights in how users interact with their website, provided that this service can also use the collected data for its own purposes. Moreover, websites may embed functionality of a social platform through which users can engage with each other. Because the resource containing embedded functionality is requested upon each page visit, the social platform can track which websites their users visit.

The main technique that is used to track users across different websites is by means of third-party cookies. More precisely, a script that is included on a wide range of websites, e.g. to display advertisements, triggers a request to the server of the tracker. Subsequently, the tracker checks whether this request contains a cookie, and either associates the triggered request with the profile of the user, or creates a new profile and responds with a `Set-Cookie` header containing the newly generated cookie. In the latter case, the user's browser will associate the cookie with the site of the tracker, and will include it in all subsequent requests to it. This allows the tracker to follow users across all websites that include a script that initiates the request to the tracker.

Because of the raised awareness of online tracking among the general public, many users delete cookies on a regular basis [12], which results in a seemingly new user profile from the tracker's perspective. As a reaction, some online trackers have resorted to more extensive tracking methods, such as respawning cookies via Flash [44] and other web mechanisms [4], and browser fingerprinting [2, 13, 44, 52]. As the evaluation presented in this paper mainly focuses on cookie policies imposed by browsers or browser extensions, our main focus is on "traditional" user tracking by means of third-party cookies. However, because the more recent tracking mechanisms also rely on sending requests to the tracker, e.g. containing the browser fingerprint, these are also subjected to the browser and extension policies. Bypasses of these policies can also be leveraged by trackers to smuggle their requests past the protection mechanisms.

3 Framework

Despite all standardization efforts, browser implementations may exhibit inconsistent behavior or even deviate from the standard. Additionally, web features from different standards may interfere with each other, causing unintended side-effects, which may affect the security and privacy guarantees. Despite prior efforts to verify these guarantees [22, 25], the real-world prevalence of inconsistencies remains hard to measure as modern browsers consist of millions of lines of code, or may be proprietary, preventing researchers access to their source code. In this paper, we evaluate the validity of constraints that are imposed on stateful third-party requests, either by browsers themselves or by browser extensions. Because of the limitations of source-code analysis, we design a framework that considers browsers, in various configurations, as a black box. This section outlines the design choices and implementation of this framework. The source code of our framework has been made publicly available.[1]

3.1 Framework design

The goal of our framework is to detect techniques that can be used to circumvent policies that strip cookies from cross-site requests, or that try to block these requests completely. To achieve this, our framework consists of various components ranging from browser control to test-case generation. These components and their interactions are depicted in Figure 2, and discussed in the following sections.

[1] `https://github.com/DistriNet/xsr-framework`

3.1.1 Browser manipulation

The framework is driven by the *Framework Manager* component, which is provided with information on which browsers and browser extensions need to be analyzed. The manager instructs the *Browser Control* component to create a specific browser instance with the predefined settings. The controller will then instruct the browser instance to visit one of the generated test-cases by leveraging browser-specific Selenium WebDriver[2] implementations. Browsers that do not have Selenium support, are controlled by manually configuring a browser profile and are then launched through the command-line.

3.1.2 Test environment

Prior to executing all test scenarios, the browser instance is first prepared. More specifically, on the target domain, i.e. the domain for which the test cases will try to initiate an illegitimate cross-site request, we install several cookies. Each of these cookies has different attributes: none, which does not impose any restrictions on the cookies, `HttpOnly`, which restricts the cookie from being accessed by client-side scripts, and `Secure`, which only allows this cookie to be sent over an encrypted connection. Throughout the remainder of the text, we refer to cookies as cookies with any one of these attributes, unless explicitly stated otherwise. Furthermore, for browsers that support it, we installed two cookies with the `SameSite` attribute: one with the value set to `lax`, and one set to `strict`. Finally, we instruct the browser to route all requests through a proxy, allowing us to capture and analyze the specific requests that were initiated as part of a test.

3.2 Test-case generation

Because of the abundance of features and APIs implemented in modern browsers, there exist a very large number of techniques that can be leveraged to trigger a cross-site request. For each such technique, our framework generates a web page containing a relevant test case.

3.2.1 Request-initiating mechanisms

As there exists no comprehensive list of all feature that may initiate a request, we leveraged the test suites from popular browser engines, such as WebKit, Firefox, as well as the web-platform-tests project by W3C[3] to compose an extensive list of different request methods. In addition, we analyzed several browser specifications to verify the completeness of this list. What follows is a

[2] `https://www.seleniumhq.org/`

[3] `https://github.com/w3c/web-platform-tests`

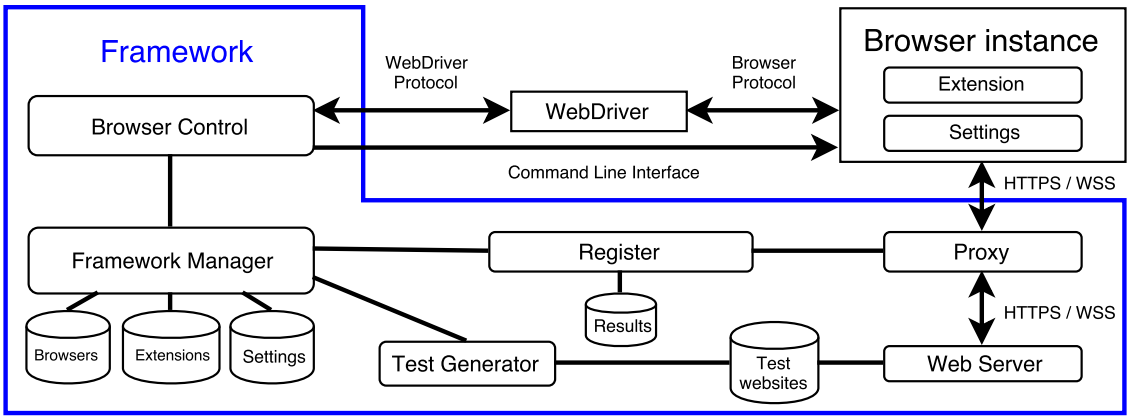

Figure 2: Design of the framework that we used to detect bypasses of imposed cross-site request policies.

summary of the mechanisms we used, subdivided into seven different categories.

HTML tags The first group of request mechanisms consists of HTML elements that can refer to an external resource, such as `<img>`, `<iframe>` or `<script>` tags. Upon parsing the HTML document, the browser will initiate requests to fetch the referred resources. As a basis, we used the HTTPLeaks project[4], which contains a list of all possible ways HTML elements can leak HTTP requests. This list was combined with techniques related to features that were recently introduced, and account for 196 unique methods. It should be noted that all HTML-based requests only initiate GET requests.

Response headers Response headers allow websites to include extra information alongside the resource that is served. We found that two classes of response headers may trigger an additional request, either as soon as the browser receives the headers or upon certain events. The first class of such response headers are `Link` headers, which indicate relationships between web resources [38]. The header can be used to improve page-load speeds by signaling to the browser which resources, such as stylesheets and associated web pages, can proactively be fetched. In most cases, the browser will request the referenced resources through a GET request.

The other class of response headers that initiate new requests are related to Content Security Policy (CSP) [1]. More precisely, through the `Content-Security-Policy` header[5], a website can, among other things, indicate which resources are allowed to be loaded. Through the `report-uri` directive, websites can indicate that any violations of this policy should be reported, via a POST request to the provided URL. Recently, another directive named `report-to` has been proposed, which allows reporting through the Reporting API [19]. As this directive and API are not yet supported by any browser, we excluded them from our analysis. Nevertheless, they are a prominent example of the continuously evolving browser ecosystem, and highlight the importance of analyzing the unexpected changes new features might bring along.

Redirects Top-level redirects are often not regarded as cross-site requests, because stripping cookies from them would cause breakage of many existing websites. Nevertheless, we included them in our evaluation for the sake of completeness, because various scenarios exist in which top-level redirects can be abused. For instance, a tracker trying to bypass browser mitigations can listen for the `blur` event on the `window` element, which indicates that the user switched tabs. When receiving this event, the tracker could trigger a redirect to its own website in the background tab, which would capture information from the user and afterwards redirect him back to the original web page. In our framework, we evaluate redirection mechanisms through the `Location` response header, via the `<meta>` tag, setting the `location.href` property and automatically submitting forms.

JavaScript Browsers offer various JavaScript APIs that can be used to send requests. For instance, the XMLHttpRequest (XHR) API can be used to asynchronously send requests to any web server [33]. More recently, the Fetch API was introduced, which offers a similar functionality and intends to replace XHR [30]. Similarly, the Beacon API can be used to asynchronously send POST requests, and is typically used to transmit analytic data as

[4] `https://github.com/cure53/HTTPLeaks`

[5] There also exists experimental CSP headers such as `X-Content-Security-Policy` and `X-WebKit-CSP`, as well as a report-only header.

it does this in a non-blocking manner and the browser ensures the request is sent before the page is unloaded [29]. Finally, there are several browser features that allow web developers to set up nonstandard HTTP connections. For instance, the WebSocket API can be used to open an interactive communication session between the browser and the server [32]. Also, the EventSource API can be used to open a unidirectional persistent connection to a web server, allowing the server to send updates to the user [34]. The latter two mechanisms are initiated using a GET request.

PDF JavaScript In addition to statically showing information, PDFs also have dynamic features that are enabled through JavaScript code embedded within the PDF file. For example, through the JavaScript code it is possible to trigger POST requests by sending form input data. The capabilities of the PDF and the JavaScript embedded within it, depend on the viewer that is used. Next to the system-specific viewer, some browsers also implement their own PDF viewer, which shows the contents in a frame. The viewer used by Chrome and Opera, PDFium [18], is implemented as a browser extension and does support sending requests. To our knowledge, this is not the case for Firefox' PDF.js library [17], as we did not manage to simulate this, nor did we find any source to confirm this.

AppCache API Although the AppCache API has been deprecated, it is still supported by most browsers [35]. This mechanism can be used to cache specific resources, such that the browser can still serve them when the network connection is lost. Web developers can specify the pages that should be cached through a manifest file. When the browser visits a page that refers to this file, the specified resources, which may be hosted at a different domain, will be requested through a GET request and subsequently cached.

Service Worker API Service workers can be seen as a replacement for the deprecated AppCache API. They function as event-driven workers that can be registered by web pages. After the registration process, all requests will pass through the worker, which can respond with a newly fetched resource or serve one from the cache. Next to fetching the requested resources, service workers can also leverage most[6] browser APIs to initiate additional requests.

[6] XMLHttpRequest is not supported in service workers.

3.2.2 Test compositions

The most straightforward way to initiate a new request is to include the mechanism directly in the top-level frame. For example, for the purpose of tracking, web developers typically include a reference to a script or image hosted at the tracker's server. However, because their top-level document can include different documents through frames, it is possible to create more advanced test compositions. In our framework, we tested 8 test-case compositions, where resources from different domains were included in each other, either through an `<iframe>` or by specific methods, such as `importScripts` in JavaScript. As we did not detect any behavior related to the test-case compositions, we omit the details from the paper. We refer to Appendix A for an overview of the different compositions that were used.

3.3 Supported browser instances

In order to generalize our results, and detect inconsistencies we evaluated a wide variety of browser configurations. These configurations range over the different browsers and their extensions, considering all the relevant settings.

3.3.1 Web browsers

The primary goal of our evaluation was to analyze browsers for which inconsistencies and bypasses would have the largest impact. On the one hand, we included the most popular and widely used browsers: Chrome, Opera, Firefox, Safari and Edge. On the other hand, we also incorporated browsers that are specifically targeted towards privacy-aware users, and thus impose different rules on authenticated third-party requests. For instance, Tor Browser makes use of double-keyed cookies: instead of associating a cookie with a single domain, the cookies are associated with both the domain of the top-level document as well as domain that set the cookie. For example, when siteA.com includes a resource from siteB.com that sets a cookie, this cookie will not be included when siteC.com would include a resource from siteB.com. Finally, we also included the Cliqz browser, which has integrated privacy protection that is enforced by blocking requests to trackers.

3.3.2 Browser settings

Most modern web browsers provide an option to block third-party cookies. While this can be considered as a very robust protection against both cross-site attacks and third-party tracking, it may also interfere with the essential functionality for websites that rely on cross-site communication. Moreover, some browsers provide built-in

functionality to prevent requests from leaking privacy-sensitive information. For instance, Opera offers a built-in ad blocker that is based on blacklists. By default, the anti-tracking and ad blocking lists from EasyList and EasyPrivacy are used, but users are able to also define custom ones. In our framework, we only considered the default setting of the built-in protection. Another browser that provides built-in tracking protection is Firefox. Here, the mechanism is enabled by default when browsing in "Private mode", and also leverages publicly available and curated blacklists [23].

Recently, Safari introduced its own built-in tracking protection, which uses machine learning algorithms to determine the blacklist [49]. Requests sent to websites on this blacklist are subjected to cookie partitioning and other measures to prevent the user from being tracked. For example, cookies will only be included in a cross-site request when there was a first-party interaction within the last 24 hours with the associated domain. Although we were unable to infer the rules of these machine learning algorithms, we still subjected this built-in option to our framework in order to be complete.

3.3.3 Browser extensions

Next to built-in tracking prevention, users may also resort to extensions to prevent their browsing behavior and personal information from leaking to third parties. As these extensions may also impose restrictions on how requests are sent, and whether cookies should be sent along in third-party requests, we also included various anti-tracking and ad blocking extensions. Due to the excessive amount of such extensions, we were unable to test all. Instead, we made a selection based on the extension's popularity, i.e. the total number of downloads or active users, as reported by the extension store. In total, we evaluated 46 different extensions for the 4 most popular browsers (Chrome, Opera, Firefox and Edge). An overview of all extensions that were evaluated can be found in Appendix B.

Most browsers' anti-tracking and ad blocking extensions share a common functionality. By making use of the WebRequest API [31], extensions can inspect all requests that are initiated by the browser. The extension can then determine how the request should be handled: either it is passed through unmodified, or cookies are removed from the request, or the request is blocked entirely. This decision is typically made based on information about the requests, namely whether it is sent in a third-party context, which element initiated it, and most importantly, whether it should be blocked according to the block list that is used. It should be noted that for the browser extension to work correctly, it should be able to intercept *all* requests in order to provide the promised guarantees. This is exactly what we evaluate by means of our framework.

4 Results

By leveraging our framework that was introduced in Section 3, we evaluated whether it was possible to bypass the policies imposed on third-party requests by either browsers or one of their extensions. The results are summarized in Table 1, Table 2, and Table 3, and will be discussed in more detail in the remainder of this section. These three tables follow a similar structure. For each category of request-triggering mechanism, we indicate whether a cookie-bearing request was made for at least one technique within this category using a full circle (●). A half circle (◐) indicates that for at least one technique within that category a request was made, but that in all cases all cookies were omitted from the request. Finally, an empty circle (○) indicates that none of the techniques of that category managed to initiate a request. Note that these results only reflect regular, `HttpOnly` and `Secure` cookies. Same-site cookies are discussed in Section 4.3. We refer to a more detailed explanation about the bug reporting in Appendix C through the indicated [bug#] tags. For a more detailed view of detected leaks and leaks for future browser and extension versions, we kindly direct you to our website.[7]

4.1 Web browsers and built-in protection

The results of applying our framework to the 7 evaluated browsers, both with their default settings as with the built-in measures that aim to prevent online tracking enabled, are outlined in Table 1. All tests are performed on the browser versions mentioned in this table, unless stated otherwise. In general, it can be seen that differences in browser implementations, often lead to differences in results. The most relevant results are discussed in more detail in the following sections.

4.1.1 Default settings

Under default configuration, nearly all of the most widely used browsers send along cookies with all third-party requests. Exceptionally, due to enabling its tracking protection by default, Safari only does so for redirects. We will discuss this further in Section 4.1.3 with the other evaluated built-in options.

Besides Safari, the privacy-oriented browsers also generally perform better in this regard: with a few exceptions, both Cliqz and Tor Browser manage to exclude cookies from all third-party requests. Most likely because redirects are not considered as cross-site (as the

[7] `https://WhoLeftOpenTheCookieJar.com`

domain of the document changes to that of the page it is redirected to), cookies are not excluded for redirects. However, as we outlined in Section 3.2, this technique could still be used to track users under certain conditions.

```
<img src="data:image/svg+xml,
<svg>
<image xlink:href='https://
third-party.com/leak'>
</image>
</svg>">
```

Listing 1: Bypass technique found for Cliqz

Another interesting finding is that in the HTML category, we found that for several mechanisms Cliqz would still send along cookies with the third-party request. An example of such a mechanism is shown in Listing 1. Here an `<img>` element included an SVG via the `data:` URL. Possibly, this caused a confusion in the browser engine which prevented the cookies from being stripped.

4.1.2 Third-party cookie blocking

In addition to the default settings, we also evaluated browsers when these were instructed to block all third-party cookies. For Tor Browser, this feature was already enabled by default. Consequently, Table 1 contains no results for Tor Browser under these settings.

Similar to what could be seen from the results of the privacy-oriented browsers, top-level redirects are not considered as third-party, and thus do not prevent a cookie to be sent along with the request. One of the most surprising results is that the browsers that use the PDFium reader to render PDFs directly in the browser (Google Chrome and Opera), would still include cookies for third-party requests that are initiated from JavaScript embedded within PDFs [bug1]. Because PDFs can be included in iframes, and thus made invisible to the end user, and because it can be used to send authenticated POST requests, this bypass technique could be used to track users or perform cross-site attacks without raising the attention of the victim. This violates the expectations of the victim, who presumed no third-party cookies could be included, which should safeguard him completely from cross-site attacks. At the time of writing, PDFium only provides support for sending requests, but does not capture any information about the response. As such, XSSI and cross-site timing attacks are currently not possible. However, as indicated in the source code[8], this functionality is planned to be added.

Because the option to block third-party cookies was removed from the latest Safari, we had to use a previous version (Safari 10). We found that setting cookies in a third-party context was successfully blocked. However, cookies - set in a first-party context - were still included in cross-site requests [bug2]. On top of that, we also found that Safari's option to block all cookies suffered from somewhat the same problem. Likewise, it managed to block the setting of third-party cookies, but cookies that were set before enabling this option were still included in cross-site requests. This problem was solved in Safari 11 by deleting all cookies upon enabling the option to block all cookies.

For Edge, we found that, surprisingly, the option to block third-party cookies had no effect: all cookies that were sent in the instance with default settings, were also sent in the instance with custom settings [bug3]. We believe that this may have been the result of a regression bug in the browser, which disabled support for this feature but did not remove the setting.

4.1.3 Built-in protection mechanisms

In total, we evaluated three built-in mechanisms that protect against tracking (Firefox' and Safari's tracking protection mode), or block advertisements (Opera's ad blocker). For Firefox and Opera, our framework managed to detect several bypasses. Although Opera's ad blocker managed to block all requests that were triggered by headers or by JavaScript embedded in PDFs, in all other categories cookie-bearing requests were made [bug4]. Although it did manage to block certain requests, e.g. for HTML tags, out of the 58 requests that were sent in the regular browsing mode, 6 were not blocked. These 6 bypass techniques spanned different browser mechanisms (CSS, SVG, `<input>` and video), so it is unclear why these are treated differently.

For Firefox, we observed comparable results: although many requests were blocked (e.g. for the HTML category, 46 out of 51 requests were blocked), for each applicable category there was at least one technique that could circumvent the tracking protection [bug5]. By analyzing the Firefox source code, we traced the cause of these bypasses back to inconsistencies in the implementation. We discuss this in more detail in Section 6.1.

In contrast to the former built-in options, Safari's Intelligent Tracking Prevention managed to mitigate all third-party cookies to a tracking domain, apart from redirects. However, we found that future completeness can be undermined by having this option disabled for even a short interval. Third-party cookies set in this interval by tracking domains, which otherwise would have been prevented, will still be included in cross-site requests after enabling the option again, identical to the results when the option is disabled. Luckily, this option is enabled by default, so future completeness can only be affected through explicit disabling by the user. As we already

[8] `https://chromium.googlesource.com/chromium/src/+/66.0.3343.2/pdf/out_of_process_instance.cc#1437`

	AppCache	HTML	Headers	Redirects	PDF JS	JavaScript	SW
Chrome 63	●	●	●	●	●	●	●
- Block third-party cookies	◐	◐	◐	●	●	◐	◐
Opera 51	●	●	●	●	●	●	●
- Block third-party cookies*	◐	◐	◐	●	●	◐	◐
- Ad Blocker	●	●	○	●	○	●	●
Firefox 57	●	●	●	●	○	●	●
- Block third-party cookies	◐	◐	◐	●	○	◐	◐
- Tracking Protection	●	●	●	●	○	●	●
Safari 11	○†	◐	○	●	○	◐	N/A
- No Intelligent Tracking Prevention	●†	●	○	●	○	●	N/A
- Block third-party cookies‡	●†	●	◐	●	○	●	N/A
Edge 40	●	●	◐	●	○	●	N/A
- Block third-party cookies	●	●	◐	●	○	●	N/A
Cliqz 1.17*	◐	●	◐	●	○	◐	◐
- Block third-party cookies	◐	◐	◐	●	○	◐	◐
Tor Browser 7	○	◐	◐	●	○	◐	N/A

●: request with cookies ◐: request without cookies ○: no request

* Secure cookies were omitted in all requests.

† Safari does not permit cross-domain caching over https (only over http).

‡ Safari 10.1.2

Table 1: Results from the analysis of browsers and their built-in security and privacy countermeasures.

mentioned in Section 3.3.2, third-party cookies will be included if first-party interaction has been occurred in the last 24 hours. This can be provoked by redirects or pop-ups to the tracking domain, although pop-ups are blocked by default.

4.2 Browser Extensions

In total, we evaluated 31 ad blocking and 15 tracking protection extensions. The results are summarized in Table 2 and Table 3 respectively. Due to space constraints, we aggregated extensions in different sets when these shared the same category-level results. Note that within a single set, extensions may still exhibit different results within one category. An overview of all browser extensions that were considered can be found in Appendix B. Guided by the resulting data, we found several common causes for the discovered bypasses.

Considering the results of all Chrome- and Opera-based extensions, it is clear that none of these managed to block the cookie-bearing third-party request when the request is initiated by JavaScript code embedded within a PDF. Although this result is similar to the results we observed when the browser was instructed to block all third-party cookies, the specific cause slightly differs. As the requests are sent from within a browser extension, the browser does not regard it as a cross-site request, and thus does not strip its cookies in the case when the "block third-party cookies" setting is enabled. However, another issue arises when a browser extension wants to block these requests: the WebExtension API does not allow an extension to intercept traffic from another extension. Consequently, this issue can not be mitigated by the anti-tracking and ad blocking extension developers [bug6].

Only few browser extensions correctly block cross-site requests initiated through the AppCache API. By analyzing the source code of the bypassed extensions, we found that these shared the same root cause. Although the listener for the `onBeforeRequest` event was always able to intercept the request, the extensions verified the provided tab identifier. However, for requests that originated from AppCache, this identifier was set to `-1`, a value that was not expected by the extension, as it may also be related to inherent browser functionality such as address bar autocompletion. As extension developers try to prevent interfering with regular browsing behavior, most extensions performed no actions on requests that caused these unexpected parameters [bug8].

Furthermore, we found that for requests initiated from service workers bypasses were made possible due to the same reasons. However, in this case Firefox-based extensions did manage to block the third-party requests. We found that this is because Firefox assigns the tab identifier to the tab on which the service worker was originally registered. As a result, from the perspective of the browser extension this seemed as a regular request, thus allowing the normal policies to be applied. In to-

tal, we found that 26 browser extension policies could be bypassed with the AppCache technique, and 20 through service workers.

Contrasting to extensions of other browsers, almost every Firefox-based extension could be bypassed in the HTML category. In most cases, this was caused by a `<link>` element, which `rel` attribute was set to `"shortcut icon"`. By further analyzing this case, we traced back the cause of this issue to an implementation bug in the WebExtension API. We found that the `onBeforeRequest` event did not trigger for requests originating from this link element [bug7]. Although abusing this bug may not be straightforward, as it is only sent when a web page is visited for the first time, it does indicate that browsers exhibit small inconsistencies, which may often lead to unintended behavior.

In the JavaScript category, we found that most extensions could be bypassed with at least one technique: for the tracking extensions, only a single extension managed to block requests initiated by JavaScript. Most prevalently, a bypass was made possible because of WebSocket connections. We found that a common mistake extension developers made, was in the registration on the `onBeforeRequest` event. The bypassed extensions set the filter value to `[http://*/*, https://*/*]`, which would allow intercepting all HTTP requests, but not WebSockets, which use the `ws://` or `wss://` protocol [bug8]. Hence, to be able to intercept all requests, the filter should include these protocols or use `<all_urls>`. Of course, the configuration of the manifest file should be updated accordingly.

In summary, we found that for every built-in browser protection as well as for every anti-tracking and ad blocking browser extension, there exists at least one technique that can bypass the imposed policies. Moreover, we found that most instances could be bypassed by using different techniques, which have different causes.

4.3 Same-site cookie

Through the tests we performed to evaluate the validity of same-site cookies, we detected incorrect behaviors for Chrome, Opera and Edge. No bugs were found for Firefox' implementation of this policy.

For Chrome and Opera, the incorrect behavior was caused by the prerendering functionality [46]. By including `<link rel="prerender" href="...">` on a web page, the visitor's browser will initiate a request to the referenced web page. If this web page resides on another domain, the resulting cross-site request will include all same-site cookies [bug9]. This bypasses the same-site cookie policy as defined by the Internet Draft; only same-site cookies in lax mode are allowed to be included.

For Edge (versions 16 and 17, which support same-site cookies), we detected similar incorrect behaviors, although caused by different functionalities [bug10]. Here, `<embed>` and `<object>` tags can be leveraged to send cross-site requests that include all same-site cookies, by pointing to another domain using the `src` and `data` attributes respectively. This also holds for requests that are sent for opening a cross-site WebSocket connection through the WebSocket API. No same-site cookies should be included at all in these requests according to the Internet Draft. On top of that, we also found that same-site cookies in `strict` mode are included in requests initiated by a variety of redirects, while this is only allowed for same-site cookies in `lax` mode. This was detected for redirects through the `<meta>` tag, `location.href` property and `Location` response header.

5 Real-world Abuse

Tracking companies and advertisers have been reported to circumvent ad blockers and anti-tracking extensions. For example, due to limitations of the WebExtension API, Pornhub managed to circumvent all ad blocking extensions by levering WebSockets [10]. As a response, several popular ad blocking extensions such as Adblock Plus and uBlock implemented a mitigation that would override the WebSocket prototype. Soon after, this mitigation was again circumvented by Pornhub, who this time leveraged WebWorkers.[9] Only when support for intercepting WebSocket connections was added to the WebExtension API, browser extensions managed to prevent Pornhub's bypasses. However, as our results show, not all browser extensions have adopted these defenses. Motivated by the seemingly strong incentives of certain trackers to circumvent request and cookie policies imposed by browser extensions, we performed an experiment to analyze whether any of the bypass techniques introduced in this paper are actively being used in the wild.

5.1 Use of bypass methods

We performed a crawl of the 10,000 most popular websites according to Alexa. For each website, we visited up to 20 pages with a Headless Chrome instance (version 64.0.3282.119, on Ubuntu 16.04), and analyzed all requests that were initiated by one of the new bypass techniques we reported in Section 4. In total, 160,059 web pages were visited by our crawler, and on each page we analyzed all third-party requests.

Next, we determined whether a cross-site request should be classified as tracking or advertising. To this

[9] `https://github.com/gorhill/uBlock/issues/1936`

		AppCache	HTML	Headers	Redirect	PDF JS	JavaScript	SW
Chrome	SET A1 (3/14)	●	●	●	●	●	●	●
	SET A2 (3/14)	●	○	◐	●	●	●	●
	SET A3 (1/14)	●	○	○	●	●	●	●
	SET A4 (1/14)	●	○	○	●	●	○	●
	SET A5 (1/14)	●	○	○	○	●	●	●
	SET A6 (3/14)	●	○	○	○	●	○	●
	SET A7 (2/14)	○	○	○	●	●	○	○
Opera	SET A8 (2/9)	●	●	●	●	●	●	●
	SET A9 (1/9)	●	○	◐	●	●	●	●
	SET A10 (2/9)	●	○	○	●	●	●	●
	SET A11 (1/9)	●	○	○	●	●	○	●
	SET A12 (1/9)	●	○	○	○	●	●	●
	SET A13 (1/9)	●	○	○	○	●	○	●
	SET A14 (1/9)	○	○	○	●	●	○	○
Firefox	SET A15 (2/5)	●	●	◐	●	○	●	○
	SET A16 (1/5)	●	●	○	●	○	○	○
	SET A17 (1/5)	●	●	○	○	○	○	○
	SET A18 (1/5)	○	●	○	●	○	○	○
Edge	SET A19 (1/4)	●	●	◐	●	○	●	N/A
	SET A20 (1/4)	●	○	○	●	○	●	N/A
	SET A21 (1/4)	○	●	○	●	○	●	N/A
	SET A22 (1/4)	○	○	○	●	○	●	N/A

●: request with cookies ◐: request without cookies ○: no request

Table 2: Results from the analysis of ad blocking extensions per browser.

purpose, we used the EasyList and EasyPrivacy lists[10] which contain regular expressions used by various popular browser extensions to determine whether requests should be blocked. In Table 4, we show the number of unique tracking or advertising domains, that make use of one of the bypass techniques that we found to be most successful. We only count the second-level domain name of the tracker or advertiser to whom the request was sent.

To evaluate whether the advertising or tracking host leveraged one of the techniques to purposely circumvent browser extensions, we visited the web pages on which these trackers or advertisers were included. For each page visit, we enabled the browser extension that may be bypassed with the detected technique. We found that all uses of the methods were legitimate, and the requests to the trackers and advertisers were never initiated because either the script or frame containing the bypass functionality was preemptively blocked. Although we did not encounter any intentional abuse in the 10,000 websites we analyzed, it is possible that trackers may actively try to avoid detection, for instance by only triggering requests upon human interaction. Moreover, as there exists a very wide spectrum of advertisers and trackers, some of these may not have been present in our dataset.

5.2 Evaluating unknown techniques

In order to evaluate whether any bypass technique was used that was not detected by our framework, we compared the DNS traffic generated by every of the 160,059 visited web pages with the requests that we could detect from each visit. More precisely, we ran every browser instance in a separate Linux namespace and used tcpdump to capture all DNS requests the browser generated. Next, we aggregated all DNS requests that could not be traced back to a captured request and used an aggregated list[11] to mark those directed towards trackers and advertisers. These DNS requests could be indicative of a bypass technique we were previously unaware of.

The preliminary analysis of this data indicated that 4,701 web pages triggered DNS requests for which we did not capture any HTTP request. However, we found that in most cases new resources were still being loaded when we closed the web page (15 seconds after opening it). We re-evaluated these web pages but now allowed the browser 120 seconds to finish loading all resources. This resulted in 865 web pages that triggered a non-corresponding DNS request to a total of 77 different hosts. A manual analysis of these showed that the vast majority was due to DNS prefetching and the remainder was still caused by requests that were interrupted when closing the browser. These results indicate the completeness of our framework, as we did not find any bypass technique that our framework was unable to detect.

[10] https://easylist.to/

[11] https://github.com/notracking/hosts-blocklists

		AppCache	HTML	Headers	Redirect	PDF JS	JavaScript	SW
Chrome	SET B1 (1/6)	●	●	●	●	●	●	●
	SET B2 (1/6)	●	●	●	○	●	●	●
	SET B3 (3/6)	●	○	○	●	●	●	●
	SET B4 (1/6)	●	○	○	○	●	○	●
Opera	SET B5 (1/4)	●	●	●	●	●	●	●
	SET B6 (2/4)	●	○	○	●	●	●	●
	SET B7 (1/4)	●	○	○	○	●	●	●
Firefox	SET B8 (1/4)	●	●	●	●	○	●	●
	SET B9 (1/4)	●	●	○	●	○	●	○
	SET B10 (1/4)	●	●	○	○	○	●	○
	SET B11 (1/4)	◐	◐	◐	●	○	●	●
Edge	SET B12 (1/1)	●	●	○	●	○	●	N/A

●: request with cookies ◐: request without cookies ○: no request

Table 3: Results from the analysis of tracking protection extensions per browser.

Category	Technique	Tracking domains	Advertising domains
AppCache	`CACHE:`	0	1
Header	`Link: <url>; rel=next`	0	0
	`Link: <url>; rel=prefetch`	0	1
	`CSP: report-uri: url`	8	1
JS	`sendBeacon(url)`	56	18
	`new WebSocket(url)`	27	7
HTML	`<link rel="shortcut icon"`	4	10
	`<link rel=apple-touch-icon`	0	2
	`<img srcset="url">`	0	3

Table 4: Unique number of tracking or advertising domains that make use of one of the potential bypass techniques

6 Discussion

As we have shown in Section 4, through our framework, which evaluated several browsers and browser extensions in various configurations, we uncovered numerous instances where an authenticated third-party request could circumvent the imposed restrictions. We found that this unintended behavior can be traced back to several factors, which can be classified as implementation errors, misconfiguration and design flaws. In this section, we discuss which measures can be taken to remedy the discovered circumventions.

6.1 Browser implementations

Most of the browsers that we evaluated have built-in support for suppressing cookies of third-party requests. Our results show that only the Gecko-based browsers (Firefox, Cliqz and Tor Browser) manage to do this successfully. Surprisingly, we found that the blocking of third-party cookies feature in Edge had no effect. We believe that this is due to an oversight from the browser developers or a regression bug introduced when new functionality was added.

For the Chromium-based browsers (Google Chrome and Opera), we found that because of the built-in PDF reader, an adversary or tracker can still initiate authenticated requests to third-parties. Because the request is triggered from within the extension, different directives apply, thus allowing cookies to be attached. A possible mitigation for this particular issue could be to disable the functionality of triggering requests from within PDFium. However, this behavior is not unique to PDFium, and other browser extensions may also be exploited in order to send arbitrary third-party requests that bypass imposed cookie policies. As such, we propose that browsers strip cookies from all requests initiated by extensions as a default policy. As this may interfere with the operations of certain extensions, exclusions should be made possible, for instance by defining a list of cookie-enabled domains in the extension manifest.

Next to blocking third-party cookies, we also analyzed the built-in tracking protection for Firefox. Interestingly, we found that for each category of mechanisms that may trigger requests, excluding JavaScript in PDFs, there exists at least one technique that can bypass the built-in tracking protection. A manual analysis of the Firefox source code showed that these bypasses are caused by the retroactive manner in which tracking protection is implemented. More specifically, although the request-validation mechanism is applied in a central location, the validation process is only triggered when a specific flag is set, which requires modifications to every functionality that may trigger requests. While Mozilla is already aware[12] of some of the bypasses we uncovered and is working to mitigate these, we believe that our framework will assist in identifying bypass techniques, even when these are difficult to detect from the millions of lines of code.

[12] `https://bugzilla.mozilla.org/show_bug.cgi?id=1207775`

6.2 Browser extensions

For anti-tracking extensions and ad blockers, it is crucial that *all* requests can be intercepted and blocked or altered. From the results, summarized in Table 2 and Table 3, it is clear that in the current state this is not the case. In fact, we found that for every analyzed browser extension there exists at least one technique that can be used to circumvent the extension to send an authenticated third-party request. Moreover, we found that the results of the evaluated browser extensions are very disparate, even for extensions that target the same browser. For instance, out of the 15 ad blocking extensions for Google Chrome, at most 3 exhibited a similar behavior.

In part, the disparity of results can be explained by the frequent introduction of new features to browsers, which may affect the WebExtension API or cause unforeseen effects. For instance, support for intercepting WebSockets in browser exceptions was only added years after the feature became available, and after it had actively been exploited to circumvent ad blockers [11]. Furthermore, AppCache caused one of the parameters of the `onBeforeRequest` API to exhibit a different behavior, which was unexpected by most browser extensions. As a result, requests triggered by AppCache managed to bypass the vast majority of browser extensions. The same change was introduced to Chromium-based browsers when Service Workers were implemented. As a result, most extensions for Chrome and Opera can be circumvented by triggering requests from Service Workers, whereas all extensions Firefox successfully block these third-party requests. This shows that adding new features to a browser may have unforeseen side-effects on the extensions that rely on the provided APIs.

When new browser features are proposed and implemented, test cases that include the new functionality can be added to our framework, allowing browser vendors and extension developers to automatically detect and possibly mitigate unforeseen side-effects. Moreover, because all anti-tracking and ad blocking browser extensions share a common core functionality (namely, intercepting and altering or blocking requests), we propose that all these extensions use a specifically purposed API that is actively maintained. Driven by the high popularity of these browser extensions, this API could be added to the WebExtension API. Alternatively, this API could be offered in the form of an extension module, which of course needs to be maintained and requires all browser extensions to update this module.

7 Related Work

Policy implementation inconsistencies Multiple studies have shown that browser implementations often exhibit inconsistencies concerning security or privacy policies. Aggarwal et al. [3] discovered privacy violations for private browsing implementations of modern browsers through both manual and automatic analysis. On top of that, they showed that browser extensions and plug-ins can invalidate the privacy guarantees of private browsing. Schwenk et al. [41] implemented a web application that automatically evaluates the SOP implementation of browsers. In that regard, they showed that browser behaviors differ due to the lack of a formal specification. Singh et al. [43] pointed out the incoherencies in web browser access control policies. In an effort to help browser vendors find the balance between keeping incoherency-confirming features and the breakage of websites as a consequence of removing them, they developed a measurement system. Jackson and Barth [21], too, showed that newly shipped browser features can undermine existing security policies. In particular, they discuss features affected by origin contamination and propose three approaches to prevent vulnerabilities caused by the introduction of these features. Zheng et al. [55] question the integrity of cookies by revealing cookie injection vulnerabilities for major sites like those of Google and Bank of America. They showed that implementation inconsistencies in browsers can aggravate these vulnerabilities.

Ad blocking circumventions Iqbal et al. [20] examined methods that are used to circumvent ad blocking in the wild. They discuss the limitations of anti-adblock filter lists and proposed a machine learning approach to identify ad block bypasses. Storey at al. [45] also proposed new approaches to ad blocking, countering the existing flaws of traditional ad blocking methods. Their new techniques include recognition of ads trough the use of visual elements, stealth ad blocking and signature-based active ad blocking.

Trackers in the wild Roesner et al. [40] performed an in-depth empirical investigation of third-party trackers. Based on the results of this investigation, they proposed a classification for third-party trackers and developed a client-side application for detecting and classifying trackers. A large-scale crawl was performed by Englehardt and Narayanan [14] to gather insights about tracking behaviors in the wild. They found that tracking protection tools such as Ghostery proved effective for blocking undesirable third-parties, except for obscure trackers.

8 Conclusion

In this work, we introduce a framework that is able to perform an automated and comprehensive evaluation of cross-site countermeasures and anti-tracking policy implementations. By evaluating 7 browsers and 46 browser extensions, we find that virtually every browser- or extension-enforced policy can be bypassed. We traced

back the origin of these bypasses to a variety of different causes. For instance, we found that same-site cookies could still be attached to cross-site requests by levering the prerendering functionality, which did not take these policies correctly into account.

Furthermore, a design flaw in Chromium-based browsers enabled a bypass for both the built-in third-party cookie blocking option and tracking protection provided by extensions. Through JavaScript embedded in PDFs, which are rendered by a browser extension, cookie-bearing POST requests can be sent to other domains, regardless of the imposed policies. Additionally, we discovered that not every implementation of the WebExtension API guarantees interception of every request. This makes it impossible for extension developers to be completely thorough in blocking or modifying undesirable requests.

Overall, we found that browser implementations exhibited a highly inconsistent behavior with regard to enforcing policies on third-party requests, resulting in a high number of bypasses. This demonstrates the need for browsers, which continuously add new features, to be thoroughly evaluated.

The results of this research suggest that policy implementations are prone to inconsistencies. That is why we think that, as future research, the framework could be extended to evaluate other policy implementations (e.g. LocalStorage API [28], Content Security Policy [1]). In addition to that, the evaluation of mobile browsers could also be an interesting direction. This includes the mobile counterparts of major browsers for iOS and Android, but also mobile exclusives like Firefox Focus [36].

Acknowledgements

We would like to thank the reviewers for their insightful comments. This research is partially funded by the Research Fund KU Leuven.

References

[1] Content security policy level 3. W3C working draft, W3C, Sept. 2016. https://www.w3.org/TR/2016/WD-CSP3-20160913/.

[2] ACAR, G., EUBANK, C., ENGLEHARDT, S., JUAREZ, M., NARAYANAN, A., AND DIAZ, C. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14* (2014), 674–689.

[3] AGGARWAL, G., BURSZTEIN, E., JACKSON, C., AND BONEH, D. An analysis of private browsing modes in modern browsers. In *Proceedings of the 19th USENIX Conference on Security* (Berkeley, CA, USA, 2010), USENIX Security'10, USENIX Association, pp. 6–6.

[4] AYENSON, M., WAMBACH, D., SOLTANI, A., GOOD, N., AND HOOFNAGLE, C. Flash cookies and privacy II: Now with HTML5 and ETag respawning.

[5] BARTH, A. HTTP State Management Mechanism. RFC 6265, RFC Editor, April 2011.

[6] BARTH, A., JACKSON, C., AND MITCHELL, J. C. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2008), CCS '08, ACM, pp. 75–88.

[7] BLOG, M. Firefox now offers a more private browsing experience. `https://blog.mozilla.org/blog/2015/11/03/firefox-now-offers-a-more-private-browsing-experience/`, 2015.

[8] BLOG, M. S. Supporting same-site cookies in firefox 60. `https://blog.mozilla.org/security/2018/04/24/same-site-cookies-in-firefox-60/`, 2018.

[9] BORTZ, A., AND BONEH, D. Exposing private information by timing web applications. In *Proceedings of the 16th International Conference on World Wide Web* (New York, NY, USA, 2007), WWW '07, ACM, pp. 621–628.

[10] BUGREPLAY. Pornhub bypasses ad blockers with WebSockets. `https://medium.com/thebugreport/pornhub-bypasses-ad-blockers-with-websockets-cedab35a8323`, 2016.

[11] CHROMIUM. chrome.webRequest.onBeforeRequest doesn't intercept WebSocket requests. `https://bugs.chromium.org/p/chromium/issues/detail?id=129353`, 2012.

[12] COMSCORE. The impact of cookie deletion on site-server and ad-server metrics in Australia, January 2011.

[13] ECKERSLEY, P. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies* (Berlin, Heidelberg, 2010), PETS'10, Springer-Verlag, pp. 1–18.

[14] ENGLEHARDT, S., AND NARAYANAN, A. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 1388–1401.

[15] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999.

[16] GELERNTER, N., AND HERZBERG, A. Cross-site search attacks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1394–1405.

[17] GITHUB. PDF.js. `https://mozilla.github.io/pdf.js/`.

[18] GOOGLE SOURCE. PDFium. `https://pdfium.googlesource.com/pdfium/`.

[19] GRIGORIK, I., AND WEST, M. Reporting API. Tech. rep., November 2017.

[20] IQBAL, U., SHAFIQ, Z., AND QIAN, Z. The ad wars: Retrospective measurement and analysis of anti-adblock filter lists. pp. 171–183.

[21] JACKSON, C., AND BARTH, A. Beware of finer-grained origins.

[22] JANG, D., TATLOCK, Z., AND LERNER, S. Establishing browser security guarantees through formal shim verification. In *Proceedings of the 21st USENIX conference on Security symposium* (2012), USENIX Association, pp. 8–8.

[23] KONTAXIS, G., AND CHEW, M. Tracking Protection in Firefox For Privacy and Performance. *In IEEE Web 2.0 Security & Privacy* (2015).

[24] LEKIES, S., STOCK, B., WENTZEL, M., AND JOHNS, M. The unexpected dangers of dynamic javascript. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., 2015), USENIX Association, pp. 723–735.

[25] LERNER, B. S., ELBERTY, L., POOLE, N., AND KRISHNAMURTHI, S. Verifying web browser extensions compliance with private-browsing mode. In *European Symposium on Research in Computer Security* (2013), Springer, pp. 57–74.

[26] MAYER, J. R., AND MITCHELL, J. C. Third-party web tracking: Policy and technology. In *2012 IEEE Symposium on Security and Privacy* (May 2012), pp. 413–427.

[27] MICROSOFT. Platform status. `https://developer.microsoft.com/en-us/microsoft-edge/platform/status/samesitecookies/`, 2018.

[28] MOZILLA DEVELOPER NETWORK. LocalStorage. `https://developer.mozilla.org/en-US/docs/Web/API/Storage/LocalStorage`.

[29] MOZILLA DEVELOPER NETWORK. Beacon API. `https://developer.mozilla.org/en-US/docs/Web/API/Beacon_API`, 2017.

[30] MOZILLA DEVELOPER NETWORK. Fetch API. `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API`, 2017.

[31] MOZILLA DEVELOPER NETWORK. webRequest. `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/API/webRequest`, 2017.

[32] MOZILLA DEVELOPER NETWORK. WebSocket. `https://developer.mozilla.org/en-US/docs/Web/API/WebSocket`, 2017.

[33] MOZILLA DEVELOPER NETWORK. XMLHttpRequest. `https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest`, 2017.

[34] MOZILLA DEVELOPER NETWORK. EventSource. `https://developer.mozilla.org/en-US/docs/Web/API/EventSource`, 2018.

[35] MOZILLA DEVELOPER NETWORK. Using the application cache. `https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache`, 2018.

[36] MOZILLA SUPPORT. Firefox Focus. `https://support.mozilla.org/en-US/products/focus-firefox`.

[37] MOZILLA WIKI. `https://wiki.mozilla.org/Security/Safe_Browsing`.

[38] NOTTINGHAM, M. Web linking. RFC 5988, RFC Editor, October 2010.

[39] PIETRASZAK, M. Browser extensions. Draft community group report, W3C, July 2017. https://browserext.github.io/browserext/.

[40] ROESNER, F., KOHNO, T., AND WETHERALL, D. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 12–12.

[41] SCHWENK, J., NIEMIETZ, M., AND MAINKA, C. Same-origin policy: Evaluation in modern browsers. In *26th USENIX Security Symposium (USENIX Security 17)* (Vancouver, BC, 2017), USENIX Association, pp. 713–727.

[42] SHARMA, R. Preventing cross-site attacks using same-site cookies. `https://blogs.dropbox.com/tech/2017/03/preventing-cross-site-attacks-using-same-site-cookies/`, 2017.

[43] SINGH, K., MOSHCHUK, A., WANG, H. J., AND LEE, W. On the incoherencies in web browser access control policies. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2010), SP '10, IEEE Computer Society, pp. 463–478.

[44] SOLTANI, A., CANTY, S., MAYO, Q., THOMAS, L., AND HOOFNAGLE, C. J. Flash cookies and privacy. In *AAAI spring symposium: intelligent information privacy management* (2010), vol. 2010, pp. 158–163.

[45] STOREY, G., REISMAN, D., MAYER, J., AND NARAYANAN, A. The future of ad blocking: An analytical framework and new techniques.

[46] THE CHROMIUM PROJECTS. Chrome Prerendering. `https://www.chromium.org/developers/design-documents/prerender`, 2011.

[47] VAN GOETHEM, T., CHEN, P., NIKIFORAKIS, N., DESMET, L., AND JOOSEN, W. Large-scale security analysis of the web: Challenges and findings. In *International Conference on Trust and Trustworthy Computing* (2014), Springer, pp. 110–126.

[48] VAN GOETHEM, T., JOOSEN, W., AND NIKIFORAKIS, N. The clock is still ticking: Timing attacks in the modern web. In *ACM Conference on Computer and Communications Security* (2015).

[49] WEBKIT. Intelligent Tracking Prevention. `https://webkit.org/blog/7675/intelligent-tracking-prevention/`.

[50] WEST, M. 'samesite' cookie attribute. `https://www.chromestatus.com/feature/4672634709082112`, 2017.

[51] WEST, M., AND GOODWIN, M. Same-site cookies. Internet-Draft draft-ietf-httpbis-cookie-same-site-00, IETF Secretariat, June 2016.

[52] YEN, T.-F., XIE, Y., YU, F., YU, R. P., AND ABADI, M. Host fingerprinting and tracking on the web:privacy and security implications. In *The 19th Annual Network and Distributed System Security Symposium (NDSS) 2012* (February 2012), Internet Society.

[53] YU, Z., MACBETH, S., MODI, K., AND PUJOL, J. M. Tracking the trackers. In *Proceedings of the 25th International Conference on World Wide Web* (Republic and Canton of Geneva, Switzerland, 2016), WWW '16, International World Wide Web Conferences Steering Committee, pp. 121–132.

[54] ZELLER, W. P., AND FELTEN, E. W. Cross-site request forgeries: Exploitation and prevention.

[55] ZHENG, X., JIANG, J., LIANG, J., DUAN, H., CHEN, S., WAN, T., AND WEAVER, N. Cookies lack integrity: Real-world implications. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., 2015), USENIX Association, pp. 707–721.

Appendix

A Test compositions

In this section, we explicate the various test compositions that we have integrated in our framework. These compositions are shown in Table 5, together with the illustrated domains.

B Extension set population

In this section, we present the extension set populations. For the ad tracking protection extensions, these are shown in Table 6 and for the ad blocking extensions in Table 7. All extensions for Chrome, Opera and Firefox were selected based on relevant search criteria and

ID	Test composition
1	● —includes→ ●
2	● —includes→ ● —includes→ ●
3	● —includes→ ● —includes→ ●
4	● —includes→ ● —includes→ ●
5	● —includes→ ● —includes→ ● —includes→ ●
6	file:// —includes→ ●
7	● —includes→ * ●
8	● —includes→ ● —includes→ ● —includes→ ● —includes→ ●

* Iframe constructed through `data:text/html`.

Table 5: Test compositions supported by our framework.

a minimum number of users or downloads (whichever was available). Due to the unavailability of both numbers for Edge extensions, we selected Edge extensions based on the popularity of their counterparts for the other browsers. The extension "AdBlocker Lite" takes up two entries in Table 2 and 7 because we tested its two modes.

C Bug reports and responses

In this section, we address the bug reports that we filed and their subsequent responses. Bugs were reported to both browsers (Section C.1) and extensions (Section C.2). In order to not inspire any attackers or trackers, we decided to only file private bug reports. Note that bug threads mights still be private when visiting the associated link.

C.1 Built-in browser protection

[bug1] The bug that can be leveraged to bypass Chrome's and Opera's third-party cookie policy has been confirmed and is scheduled to be fixed at the time of writing. [13]

[bug2] We reported that Safari 10 does not block all third-party cookies when this option is enabled. At the time of writing, this bug has not yet been confirmed.[14]

[bug3] The bug that nullifies Edge's option to block third-party cookies has been confirmed.[15]

[bug4] The bypasses for Opera's ad blocker have been reported, however, we were not given access to the bug thread. Instead, we were given an email address through which we can inquire about the process.

[bug5] In the bug thread that we have started for bypasses concerning Firefox' tracking protection, references have been made to previously reported similar bugs that are related to Firefox' Safe Browsing feature [37].[16] For example, the AppCache API had already been reported to bypass the URL classifier used by Safe Browsing to signal websites known for phishing or malware. Although the bug has not yet been officially flagged as confirmed at the time of writing, there was an intention to fix.

C.2 Extensions

[bug6] This bug permitted cross-site requests, initiated by JavaScript embedded in a PDF, to bypass the WebExtension API in Chromium-based browsers. This made it impossible for extensions (e.g. ad blockers and anti-tracking extensions) to implement a thorough third-party cookie and request policy. Unfortunately, our bug thread was closed as WontFix,[17] because this functionality was working as intended; requests initiated by an extension (PDFium) shouldn't be interceptable by other extensions. Thread responses showed reluctance to treating PDFium differently because it would be costly and difficult to implement. We mentioned that Opera - a Chromium-based browser - actually managed to mitigate these requests

[13] `https://bugs.chromium.org/p/chromium/issues/detail?id=836746`

[14] `https://bugs.webkit.org/show_bug.cgi?id=186589`

[15] `https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/16512847`

[16] `https://bugzilla.mozilla.org/show_bug.cgi?id=1447935`

[17] `https://bugs.chromium.org/p/chromium/issues/detail?id=824705`

Set	Extension name	Version	Number of users/downloads
	Chrome Tracking Protection Extensions		
SET B1	Blur	7.7.2390	248,825 users
SET B2	ScriptSafe	1.0.9.1	286,512 users
SET B3	Ghostery	7.4.1.4	2,787,473 users
	Privacy Badger	2017.11.20	711,102 users
	Disconnect	5.18.23	918,877 users
SET B4	uMatrix	1.1.12	121,618 users
	Opera Tracking Protection Extensions		
SET B5	Blur: Protect your passwords, payments & privacy	7.7.2393	154,817 downloads
SET B6	Disconnect	5.17.5	564,628 downloads
	Privacy Badger	2017.11.20	140,381 downloads
SET B7	Ghostery	7.4.3.1	4,865,900 downloads
	Firefox Tracking Protection Extensions		
SET B8	DuckDuckGo Plus*	2017.11.30	419,351 users
SET B9	Privacy Badger	2017.11.20	411,406 users
SET B10	Ghostery Privacy Ad Blocker	7.4.1.4	1,048,907 users
SET B11	Cliqz - Schnellsuche und Trackingschutz	2.21.3	94,361 users
	Edge Tracking Protection Extensions		
SET B12	Ghostery	7.5.0.0	N/A

* Recently changed its name to "DuckDuckGo Privacy Essentials".

Table 6: Population of the tracking protection extension sets.

with its built-in ad blocker, but also proposed an alternative solution like providing a setting to block execution of JavaScript embedded in PDFs. Response to our proposition was supportive, however we are not aware of any progress on the matter. In the same bug report, we also explained the difficulties for extensions to distinct between requests initiated through the AppCache or ServiceWorker API, and requests initiated by browser functionality. However, no responses have been made in regard to this.

[bug7] We reported that requests for fetching the favicons are not interceptable through Firefox' WebExtension API and that requests initiated through the AppCache API are not easily distinguishable in Firefox. The bug thread was closed as WontFix,[18] because the first issue had already been reported and no additional effort will be made to fix the deprecated AppCache API.

[bug8] In addition to the aforementioned bugs caused through the AppCache and WebSocket API, we identified a wide variety of bugs inherent to the implementation of ad blocking and privacy protection extensions. Because of the large number of affected extensions, many without a dedicated bug tracker, we only contacted a selection of them. This selection involved the 11 most popular and recently updated extensions, most of them supported by multiple browsers, to which we reached out through a private channel. Unfortunately, only 5 extension developers responded, of which only 2 pro-actively tried and succeeded to fix the issue.

C.3 Same-site cookie

[bug9] The prerender bug that we found in Chrome and Opera has been filed through the Chromium project, where it was confirmed and scheduled to be fixed.[19]

[bug10] We have reported the several bypasses that we found for Edge's implementation of the same-site cookie policy. This bug report has been confirmed.[20]

[18] https://bugzilla.mozilla.org/show_bug.cgi?id=1447933

[19] https://bugs.chromium.org/p/chromium/issues/detail?id=709946

[20] https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/18054323/

Set	Extension name	Version	Number of users/downloads
	Chrome Ad Blocking Extensions		
SET A1	AdRemover for Google Chrome	1.1.1.0	9,463,986 users
	Windscribe - Free VPN and Ad Blocker	2.3.4	553,466 users
	uBlock	0.9.5.0	519,056 users
SET A2	AdBlocker Ultimate	2.26	628,321 users
	Ads Killer	0.99.70	2,262,911 users
	Hola ad blocker	1.21.624	143,790 users
SET A3	Fair AdBlocker	1.404	1,808,682 users
SET A4	AdGuard AdBlocker	2.7.2	4,650,713 users
SET A5	AdBlock Pro	4.3	2,134,631 users
SET A6	uBlock Adblocker Plus	2.3	332,645 users
	uBlock Origin	1.14.22	10,000,000+ users
	uBlock Plus Adblocker	1.5.2	521,915 users
SET A7	AdBlock	3.22.1	10,000,000+ users
	Adblock Plus	1.13.4	10,000,000+ users
	Opera Ad Blocking Extensions		
SET A8	AdBlocker Lite (Lite mode)	0.4.0	164,309 downloads
	AdBlock	2.57	11,199,416 downloads
SET A9	AdBlocker Ultimate	2.23	1,209,271 downloads
SET A10	Adblock Fast	1.2.0	465,483 downloads
	AdBlocker Lite (Full mode)	0.4.0	164,309 downloads
SET A11	Adguard	2.7.2	5,649,827 downloads
SET A12	ContentBlockHelper	10.2.0	371,330 downloads
SET A13	uBlock origin	1.14.16	3,738,666 downloads
SET A14	Adblock Plus	1.13.4	33,802,382 downloads
	Firefox Ad Blocking Extensions		
SET A15	AdBlock for Firefox	3.8.0	865,131 users
	AdBlocker Ultimate	2.28	448,458 users
SET A16	Adguard AdBlocker	2.7.3	299,462 users
SET A17	uBlock Origin	1.14.18	5,216,321 users
SET A18	Adblock Plus	3.0.1	13,574,386 users
	Edge Ad Blocking Extensions		
SET A19	AdBlock	2.4.0.0	N/A
SET A20	Adblock Plus	0.9.9.0	N/A
SET A21	Adguard Adblocker	2.8.4	N/A
SET A22	uBlock origin	1.14.24	N/A

Table 7: Population of the ad blocking extension sets.