One Side-Channel to Bring Them All and in the Darkness Bind Them: Associating Isolated Browsing Sessions

Tom Van Goethem
imec-DistriNet - KU Leuven
tom.vangoethem@cs.kuleuven.be

Wouter Joosen
imec-DistriNet - KU Leuven
wouter.joosen@cs.kuleuven.be

Abstract

Online tracking and fingerprinting is becoming increasingly more prevalent and pervasive. The privacy threats associated with these practices have given rise to a wide variety of privacy-enhancing solutions. However, as these solutions retroactively apply patches to existing browsers in an attempt to thwart potential attacks, it is of key importance that the complete threat surface is known such that all risks can be considered. In this paper we evaluate the browser's threat surface with regard to fingerprinting and tracking in the context of isolated browsing sessions, i.e. regular versus incognito sessions or sessions across different browsers. We uncover and evaluate three types of side-channels, and show how an adversary can exploit these to track users across sessions and even reveal the IP address of Tor users when they use a concurrent browsing session.

1 Introduction

The online advertisement industry is constantly growing. According to a report by eMarketer, digital ad spending surpassed television advertising, and is forecasted to grow to 83 billion USD [11]. In this highly competitive market, advertising companies try to attract advertisers through various means. For instance, by offering the ability to show advertisements only to a small target audience, advertisers can reach out to the people they consider most likely to buy their products. Of course, this requires the advertising company to create a profile on the users that are shown advertisements. Such a profile can be constructed by measuring the user's online activities, e.g. based on the websites that are visited, the advertiser can infer the user's interests.

By having websites include resources from the advertising company, the user's browser will trigger a request to the advertising company, and include cookies that were previously set. These third-party cookies allow the advertising company to track which websites the user visits (either based on the Referer header of the request, or by additional information provided by the including website). In a large-scale study by Libert, it was shown that 88% of the websites include third-party resources, and on 62.9%, this resulted in setting a third-party cookie [20]. These findings are a clear indicator of the growing ubiquity of online tracking.

One of the key limitations of tracking by means of third-party cookies, is that when users remove their cookies, they become an entirely new entity to the tracker. Consequently, the profile of the user that the tracker had constructed so far becomes obsolete upon removal of the cookies. To overcome this limitation, several trackers have resorted to more intrusive tracking techniques. These techniques include abusing persistent storage mechanisms, and fingerprinting the browser and device, e.g. by enumerating the fonts that are installed on the system.

As a response to the emerging[1] threat imposed by online web tracking and fingerprinting, a wide variety of privacy-enhancing efforts have been made. These range from the privacy or incognito browsing mode that is available in all popular browsers, to browser extensions such as Ghostery [14] that try to block trackers, to browsers such as Tor Browser whose main objective is to provide anonymity to their users. It has been shown that these defenses are relatively effective at defending against known fingerprinting and tracking techniques [12]. However, because these privacy-enhancing countermeasures build upon existing browser architectures, e.g. Tor Browser is based on Firefox, most defenses are applied retroactively. This underlines the importance of discovering new classes of fingerprinting techniques, as they can be used to guide existing counter-

[1] In a 2013, Acar et al. found that 0.4% of the top 1 million websites performed font enumeration in JavaScript; in a study performed 3 years later by Englehardt and Narayanan [12], the prevalence of this technique had increased almost fourfold.

measures, or inspire completely new approaches towards tackling online tracking and fingerprinting.

In this paper, we introduce three classes of fingerprinting techniques that have been little explored or are newly introduced. Moreover, we show that these techniques can be used to continue tracking users when they switch to incognito browsing mode or use a different browser.

The key contributions of our study are:

- We show that side-channel attacks on modern web browsers is a large threat surface for fingerprinting attacks.
- As a result of our analysis on various browser mechanisms, we discover 3 side-channel attacks that can be abused to track users along private browsing mode and across different browsers.
- We conclude that the proposed fingerprinting techniques can have very nefarious consequences, such as revealing the IP address of Tor users, and indicate that additional countermeasures are required to defend against the newly introduced threats.

2 Background

In the study by Krishnamurthy and Wills, the researchers perform a longitudinal analysis of online tracking between 2005 and 2008 [19]. They show that over the course of their study, tracking on the web steadily increases. Following research shows that this trend continues [23, 36]. Moreover, several researchers report that next to the tracking via cookies, an increasing number of websites are resorting to more intrusive techniques [24, 2, 33, 1]. In an attempt to significantly reduce, and ideally, completely thwart these tracking methods, various efforts have been made.

First, a number of browser extensions have been developed that aim to detect and block online trackers. A well-known example of such an extension is Ghostery, which monitors the requests and responses to all third-party resources, and uses that information to detect and block trackers. Furthermore, other extensions target a specific fingerprinting technique; for instance, the Firefox add-on CanvasBlocker[2] defends solely against canvas fingerprinting [27].

Browser vendors and developers of browser specifications are also increasingly invested in reducing the threats of fingerprinting, and try to prevent increasing the attack surface when new APIs are designed and implemented. A case in point is Firefox' "Private Browsing" mode which provides tracking protection by using the block list provided by Disconnect.me[3]. Moreover, in private mode, the browser prevents storing content originating from websites to the disk, e.g. by disabling certain APIs. At the side of specification developers, new browser APIs typically undergo a security and privacy evaluation, e.g. by W3C's Privacy Interest Group, in order to prevent enlarging the threat surface at the design level.

Finally, with a main focus on providing online anonymity, completely new browsers have been introduced, most notably Tor Browser [34]. By routing all traffic over the Tor network, Tor Browser ensures that the IP address of the user is not known to the visited website. Additionally, Tor Browser, which is based on Firefox, adds a wide variety of countermeasures against tracking, e.g. by blocking third-party cookies, and fingerprinting. These fingerprint defenses include, among other things, preventing the enumeration of fonts, and disabling various browser APIs that could potentially leak identifying information. Furthermore, as a defense-in-depth strategy, both for security as well as privacy reasons, the timers of Tor Browser have a reduced resolution (100ms), and introduce random noise.

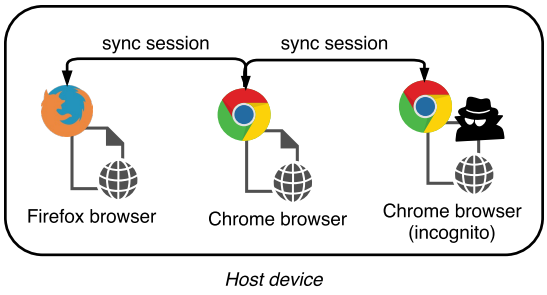

Figure 1: Threat model of cross-session user tracking.

3 Exploring Browser Operations

3.1 Threat Model

We consider a threat model as the one depicted in Figure 1. In this model, the user/victim is using multiple browsing sessions that are isolated from each other. This could either be two browsing sessions from the same browser, where one is a *regular* browsing session and the other is in *incognito* or *private* mode. Alternatively, the user could also be using different browsers to separate, and thus shield off, multiple browsing sessions from each other. The adversary, which we consider capable of running JavaScript code in the victim's browser, e.g. through a malicious advertisement, aims to associate the different sessions, and thus link the victim's identity to these multiple sessions.

In case the attack is successful, and the adversary can link together different browsing sessions, the user's ex-

[2] https://addons.mozilla.org/en-us/firefox/addon/canvasblocker/

[3] https://disconnect.me/

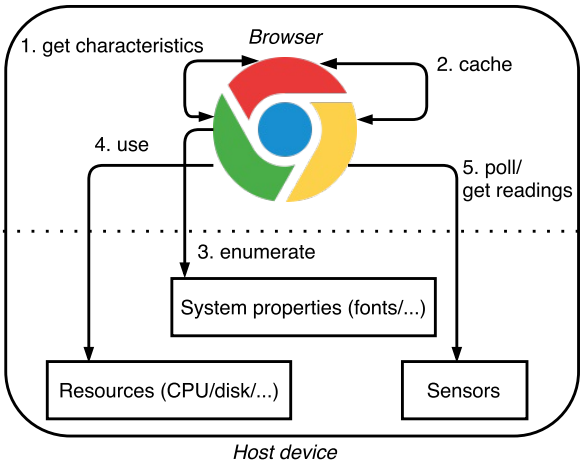

Figure 2: Overview of operations that a browser can use to interact with the host device.

pectations of privacy are broken. A concrete example of this is when users perform their day-to-day browsing in regular browsing mode, but use the incognito mode to visit sensitive websites, e.g. to look up information on certain medical conditions. An adversary can use regular tracking methods to follow the user's web visits in the regular browsing mode. However, once the user switches to incognito mode, the browser creates an entirely new environment isolated from the other browsing session, preventing the attacker to further track the user over this session. Additionally, we also consider associating isolated browsing sessions across browsers. For instance, someone could use Tor Browser to access highly sensitive information (Tor Browser provides anonymity by routing all traffic over the Tor network, and implementing numerous fingerprinting and tracking defenses [34]), but at the same time use another browser for general usage (traffic over the Tor network suffers from a high latency, and in some cases may be blocked entirely by websites). An adversary capable of associating the Tor Browser session to a browser session that is not routed over the Tor network, can reveal the IP address of the user, defeating their anonymity.

Unless an anonymity network is used, the IP address of a user will be known to all parties the user's browser requests a resource from. As was pointed out by Boda et al. [4], the IP address can be used to track users across browsers. However, this only works when the IP address is not shared among a larger group of people. As such, this tracking technique is rendered obsolete when users browse the web over a VPN, or when their traffic is tunneled through an HTTP proxy, as is common in corporate environments. The cross-browser tracking techniques introduced in this paper work irrespective of IP-level countermeasures.

Table 1: An overview of fingerprinting and tracking techniques.

Technique	Operation	Related work	Defenses	Cross-browser
Canvas FP	(4)	[27, 1]	[10], [25]†, TB	✓
Font enumeration	(3)	[9, 33, 2]	[3, 32], TB	✓
Browser properties	(1)	[22, 9]	TB	✗
System config.	(3)	[4]	TB	✓
JS Engine	(4)	[26, 31]		✗
Extension detect.	(1)	[39, 38]	IM, TB	✗
Extension detect.	(1)	Sec. 4	TB	✗
Simultaneous events	(5)	Sec. 5		✓
Covert channels	(4)	Sec. 6		✓

† Only performs detection, not a defense.
TB stands for Tor Browser, IM for incognito mode.

3.2 Methodology and Prior Findings

In order to uncover and assess techniques that can be used to link together different browsing sessions, we first make an abstraction of the browser model. In this abstraction, shown in Figure 2, we identify five conceptual operations, of which two are browser-specific, and the remaining three are based on interactions with the host device. It is important to note that because the browser-specific operations leverage information from one particular browser, these can only be used to link together *regular* and *incognito* sessions. As the other operations are performed on system-wide artifacts, which are shared among all applications, and thus by extension all browsers, these artifacts can be exploited for cross-browser session syncing.

As browser fingerprinting is a well-researched topic, a plethora of techniques have been discovered that can be used to identify users. Table 3 shows a summarized overview of fingerprinting techniques that have been previously discovered, combined with known defenses that thwart the attack. For each technique, we indicate the associated browser operation that is exploited, whether Tor Browser (TB) or incognito mode (IM) defend against the technique, and finally whether it can be used to track users across browsers. The bottom part of the table shows the tracking techniques that are discussed in the following sections. It should be noted that the extension detection technique introduced in this paper can, in contrast to prior findings, be performed in incognito mode, although extensions are disabled there by default.

Experimental setup Unless explicitly mentioned otherwise, the experiments in the following sections have been performed on a Mid-2012 MacBook Pro, with a 2.5GHz Intel Core i5 processor and 16GB RAM. These experiments were then later re-evaluated on four different devices, spanning a variety of hardware and operating systems. Unless the results from the other machines showed a different behavior, they have been omitted.

4 Enumerating Browser Extensions

Browser extensions allow users to incorporate additional features, such as blocking advertisements, into their browsers. With more than 100,000 extensions in total [21], of which the most popular one has more than 500 million downloads [41], browser extensions are an interesting target for fingerprinting. In this section, we first provide a brief view on the history of extension fingerprinting, and then introduce a novel technique, based on exploiting timing side-channels that can be used to detect the presence of browser extensions, even in incognito mode.

4.1 Previously Discovered Attacks

In 2011, James Kettle found that extensions in Chrome and Firefox could be detected by including <img> elements which source was set to the chrome:// [4] URLs of the extensions [17][5]. Since the load event of the <img> element would only fire when the extension was present, this allowed an adversary to detect all installed extensions by testing them one by one using brute force. Upon the introduction of the *Web Accessible Resources* property in the extension manifest, these techniques were largely rendered obsolete: only the resources listed by the extension developer could be accessed by external web pages. As such, the presence of extensions that did not list any resources as web-accessible could no longer be detected. In a recent study by Sjösten et al., it has been shown that despite these countermeasures, more than 50% of the 1,000 most popular Chrome extensions could still be detected (as these include at least one web-accessible resource in their manifest) [38]. It is important to note that these techniques do not work against Firefox and Safari because these generate a unique, random identifier for every installation. For Firefox, the web-accessible resources can be accessed through moz-extension://<random-UUID>/<path/to/resource> [30]; in Safari these resources are accessible under the following pattern: safari-extension://<extensionID>-<developerID> /<random-number>/<path/to/resource>. As web-accessible URLs can only be generated through the browser.extension.getURL API for Firefox, and the safari.extension.baseURI API for Safari, an adversary is unable to guess the random identifier in the extension's resource path, and therefore cannot access these resources (and thus is unable to detect its presence).

In a recent study by Starov and Nikiforakis, the researchers introduce XHOUND, a fully automated system for fingerprinting browser extensions [39]. Their system leverages the changes an extension introduces to a page's DOM. The researchers show that 9.2% of the 10,000 most popular Chrome extensions introduce detectable changes to any URL. As such, by detecting these changes, and identifying which extension caused them, the tool is capable of detecting tens of extensions in a few seconds. Interestingly, because of the intrinsic nature of this detection technique, it is not limited to a specific browser implementation, and thus can be applied to any browser that supports extensions. However, an important limitation of this technique, rendering it impractical in cross-session attacks, is that extensions are by default disabled in incognito mode. While it is possible to reenable extensions in incognito mode, it is not guaranteed that a user will enable *all* extensions there, which will therefore result in a different fingerprint. In the following section, we introduce a new technique that does not suffer from this limitation, and thus can be used to detect which extensions have been installed, even when they are disabled.

4.2 Timing Attack on Browser Extensions

Every extension following WebExtensions, a cross-browser system for developing browser add-ons, is required to define a manifest file that details various properties of the extension. The property named web_accessible_resources can be used to indicate which resources are allowed to be accessed from the web. For extensions that still have the manifest version set to 1, all resources are available by default, unless the web_accessible_resources property has been set. Contrastingly, for extensions following manifest version 2 or above, access to the extensions' resources is blocked by default, and can only be enabled by whitelisting certain resources [16].

An important observation that forms the basis for the timing attack is that the browser will always need to evaluate the extension's manifest to determine whether a requested resource can be accessed. First, the browser has to determine the default behavior: allow-by-default or whitelist-only. Next, in case only whitelisted resources are allowed, the browser has to loop over all web_accessible_resources entries (which can include wildcards) and evaluate whether any of these match the requested resource. Because these steps are only be performed on installed extensions (obviously, because for extensions that are not installed there is no manifest that can be parsed), this introduces a timing side-channel that can be abused to detect the presence of any extension. Figure 3 displays a timeline of a request to an extension's resource, showing that an early-exit for extensions that

[4] The chrome:// URL was later replaced by the chrome-extension:// URL.

[5] This technique was independently discovered a few months later by Krzysztof Kotowicz [18].

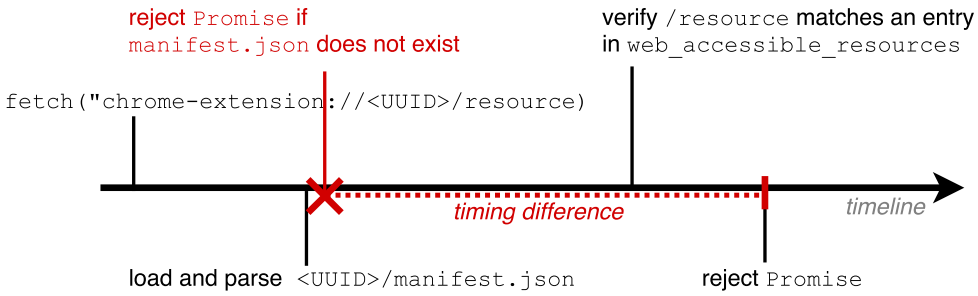

Figure 3: Timeline showing the difference in timing for accessing a resource of an installed vs. not-installed extension.

are not installed causes a timing difference.

Concretely, to determine the presence or absence of an extension, the adversary uses the Fetch API to make a request for a randomly selected endpoint of the extension, i.e. chrome-extension://<UUID>/resource, where <UUID> is the identifier of the extension, e.g. cfhdojbkjhnklbpkdaibdccddilifddb for Adblock Plus. At this point, the attacker starts a timer, e.g. by storing the result of performance.now() in a variable. After a short time period, the Promise returned by the fetch() operation will be rejected, either because the extension was not installed (and the manifest could not be evaluated), or because the requested resource was not allowed according to the web_accessible_resources property. Upon rejection of the Promise, the adversary stops the timer and records the time that was measured. Finally, the adversary obtains a similar measurement but for a resource of a non-existent extension, and compares it to the timing of the tested extension. When the measurement of the targeted extension is significantly higher, the attacker can conclude that the manifest was parsed and evaluated, and thus is present in the victim's browser.

4.2.1 Evaluation

In order to assess the success rate of this timing attack, we performed two experiments. In the first experiment, we obtained a large sample (10,000 per extension) of timing measurements, alternately for a random resource of both an installed and non-existent extension. Figure 4 shows a kernel density plot of these measurements and clearly indicates that on average, there is a noticeable timing difference between the two cases. We performed these measurements on the latest version of Google Chrome that was available at the time of writing (58.0.3029.110), on a variety of host devices, including various operating system (macOS, Linux, Windows). For most combinations, we found that results were similar to the upper graph on Figure 4.

Interestingly, we did find one exception, which exhibited a different distribution, namely on systems running Windows 10. Here, we found that after several timing measurements, the time required to perform a single measurement would significantly decrease (from 1ms to 0.5ms, and after a few hundred measurement further down to 0.25ms). We believe this behavior is related to certain optimizations at the level of the operating system or hardware (e.g. JIT compilation or caching). While this behavior might be an interesting target for fingerprinting, in line with the JavaScript performance fingerprint introduced by Mowery et al. [26], it is in fact detrimental for our extension fingerprint attack. Nevertheless, we found that by introducing a short (100ms) timing delay every 100th measurement attempt, these side-effects could largely be mitigated. The bottom graph of Figure 4 shows the distribution of timing measurements for Windows 10. This graph still shows evidence of certain performance optimizations, especially in the case of an installed extension. By increasing the timing delay, these effects can be further reduced, at the expense of a higher time required to perform the fingerprinting attack.

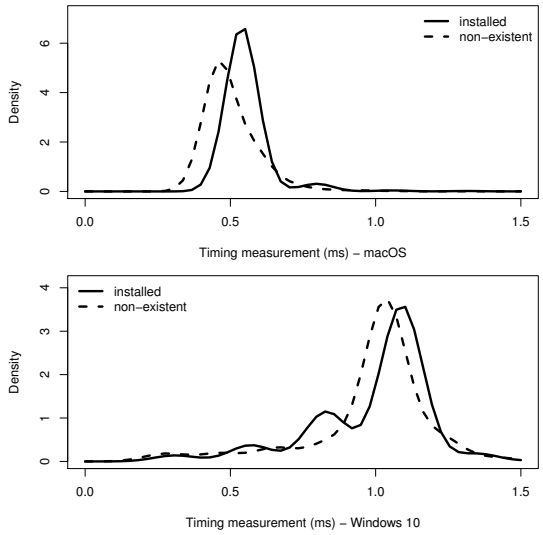

Figure 4: Distribution of timing measurements for both an installed and non-existent extension, as obtained on macOS and Windows.

Table 2: Accuracy and total execution time to detect the presence of a browser extension over a growing number of measurements.

	Detect presence		Detect absence	
Measurements	*Accuracy*	*Runtime*	*Accuracy*	*Runtime*
10	87.3%	11.36ms	88.4%	10.70ms
20	93.4%	22.67ms	93.0%	21.40ms
30	96.9%	33.44ms	97.2%	33.77ms
40	98.8%	45.97ms	97.0%	42.97ms
50	98.3%	56.61ms	99.0%	53.36ms
60	98.8%	67.27ms	98.7%	64.22ms
70	100.0%	79.63ms	99.3%	74.79ms
80	99.5%	89.67ms	100.0%	85.74ms
90	100.0%	101.86ms	100.0%	94.89ms
100	100.0%	113.62ms	100.0%	106.91ms

To further explore the feasibility of the fingerprinting attack in a real-world scenario, we performed a subsequent experiment. Because the timing measurements are not clearly distinctive, i.e. the distributions show a significant overlap, an adversary has to obtain multiple measurements before being able to determine the presence of a browser extension.

In our experiment, we set out to explore the number of measurements per extension that are required to obtain a high accuracy within an acceptable execution time. As such, we computed the average accuracy and runtime for a varying number of measurements per extension. More precisely, in the test with 10 measurements, we did a pairwise comparison[6] between the timing result of an installed and non-installed extension. Next, we counted the number of correct measurements, i.e. those for which the timing measurement was higher for the installed extension. When the number of correct measurements exceeded a predefined threshold, we consider it a successful attack. Through empirical tests, we found that 70% provides a good trade-off between correctly classifying an extension as present versus absent. Finally, we computed the average accuracy over 1,000 iterations of this test.

The results of this experiment are displayed in Table 2, and show that with around 90 measurements, the attack achieves a quasi-perfect accuracy, allowing an adversary to detect the presence or absence of an extension in a bit over 100ms. It should be noted that the reported runtime includes the measurement time for two extension identifiers (installed and non-installed, or two different non-installed extensions). As such, an adversary could reduce the execution time of his attack to approximately half of the reported values.

[6] A pairwise comparison was used to reduce the random noise that is introduced over time.

4.2.2 Defense

We have reported our findings to the Chromium team[7], and they are in the process of preventing the fingerprinting techniques. We propose two methods that can be used to prevent the attack. A first solution could be to make the identifiers of browser extensions unguessable, e.g. by making it installation-specific, as is the case with Firefox. This prevents the adversary from trying to access any extension resource, and thus completely thwarts the attack. However, a downside of this defense is that it might break existing web applications (detecting the presence of the official Chromecast extension is done by accessing a web-accessible resource[8]).

Another defense strategy could be to prevent the timing side-channel. Ideally, this should be done by guaranteeing a constant execution time when trying to access a resource of an extension. As the difference in timing between installed and non-installed extensions is relatively small (approximately 0.2ms), the operations that cause this difference could be padded to the estimated worst-case execution time (WCET) without incurring a large performance overhead.

Finally, as part of a defense-in-depth strategy, extensions that are installed, but disabled in incognito mode, should not be considered when requesting a resource from them. While this countermeasure would still allow fingerprinting browser extensions in regular browsing mode, it would prevent associating the incognito browsing sessions to the regular ones.

5 Cross-Session Events

As a result of a privacy and security review of the then-proposed Media Capture and Streams API, it was pointed out that if the devicechange event[9] would fire in all browsing contexts at the same time, this would allow websites to associate the different contexts, which may include multiple browsers or incognito mode [8]. As these events are relatively uncommon, and fire across contexts within a short timespan, an adversary can associate sessions for which the same type of event is observed within a few milliseconds of each other. As a response to these potentially nefarious consequences on the user's privacy, the specification was altered to only fire the event on tabs that were either previously given permission to access media devices, or that are active[10].

[7] https://bugs.chromium.org/p/chromium/issues/detail?id=709464

[8] https://bugs.chromium.org/p/chromium/issues/detail?id=139592#c32

[9] The devicechange event fires when a media device, such as an external microphone is attached or removed.

[10] https://github.com/w3c/mediacapture-main/pull/412/commits

Table 3: Overview of events that can be leveraged to associate browsing sessions, evaluated on Google Chrome (GC), Firefox (FF), Edge (ED). Safari (SA), and Tor Browser (TB).

Event name	Affected browsers	Incognito?	Cross-browser?
devicechange	GC, FF•, ED	✓	✓
devicemotion	GC	✓†‡	✗
languagechange	GC, FF, SA, TB	✓	✗
offline/online	GC, FF, ED, SA	✓	✓
Battery*	GC	✓	✗

• On Firefox, the devicechange event is not available, but can be simulated by constantly polling the enumerateDevices API.
† Events are only fired on active tabs.
‡ Requires a device with an accelerometer or gyroscope (e.g. Macbook Pro).
* The Battery API exposes following events: chargingchange, chargingtimechange, dischargingtimechange, levelchange

As modern browsers heavily interact with a wide variety of external components, it is possible that other APIs are subject to similar threats. In order to explore the prevalence and potential impact of related issues, we constructed a testing platform that allows us to capture and analyze a wide variety of events. Using this framework, we evaluated all events that are fired on the Window object, as well as events that can be listened on through interactions with the Navigator object. As part of this evaluation, which we performed on multiple browsers, we assessed which events could be abused to associate different browsing sessions, i.e. associating normal and incognito mode, or sessions across different browsers.

The results of our evaluation are summarized in Table 3. In total, we found 9 types of events that can be used to associate different browsing sessions; four of these are related to the Battery API, and two are indicators of connectivity. Interestingly, the events that are most effective at associating different browser sessions, even across different browsers, are the offline and online events which are fired when the host device disconnects, or reconnects, respectively. We found that the events do not fire at exactly the same time in every tab, but rather within the range of a few milliseconds. Nevertheless, because these events typically only occur rarely, the adversary can still link different browsing sessions to a single entity, even within a relatively large pool.

Another surprising finding is the one related to the languagechange event, which is fired when the user alters the default language in the browser's settings. It should be noted that the language change is not system-wide, and thus can not be used to associate cross-browser sessions. We determined that this event is fired at approximately the same time, for Google Chrome, Firefox, Safari, and the Tor Browser. In particular for the Tor Browser this could be a significant threat: by design, the Tor Browser aims to ensure unlinkability across origins, e.g. by using a different circuit for every browser tab [34]. When an event is broadcasted to all tabs at the same time, this allows an adversary to link all page visits to a single user. Although it can be considered uncommon that users would change languages in their browser, the potential consequences of this seemingly innocuous action can be relatively severe. As the Tor Browser is a privacy-oriented adaptation of Firefox, we suspect this event was overlooked (the offline and online events are disabled in order to prevent precisely these consequences).

6 Resource Contention as Covert Channel

Similar to how browsers interact with a wide variety of components of the host device, they also (have to) make use of various system resources. These resources include the CPU, e.g. to run the JavaScript code, the disk, e.g. to cache HTTP responses, the GPU, e.g. to render a web page, the device's memory, e.g. to store the contents of a JavaScript variable, ... Moreover, these resources are shared between all applications on a host device, causing these applications to contend for the shared resources.

As performance is one of the key requirements of a browser, it will completely exhaust any resource it makes use of in order to achieve the result as fast as possible. However, as resources are typically bound by an upper limit for their bandwidth, usually imposed by hardware constraints, two concurrent browser threads will not be able to use the resource at its maximum capacity. As a result, the browsers will experience a performance degradation when the resource they want to use is concurrently used by another browser (or application). This potentially introduces a covert channel which may allow an adversary to set up a communication channel between two browser sessions. These could either be within the same browser, i.e. a regular browsing session and an incognito one, or, since the resources are shared system-wide, between different browsers on the same host. The adversary could leverage this communication channel to exchange unique identifiers, thereby linking the sessions to a single entity.

In order to send information over this covert channel, the two browsers will exploit the resource contention as follows. As with all direct communication channels, there is a sender and receiver. In this case, the receiver will attempt to saturate the shared resource constantly, over the whole duration of the communication, and continuously measure its performance. The sender will first encode its message as binary, i.e. a sequence of 1's and

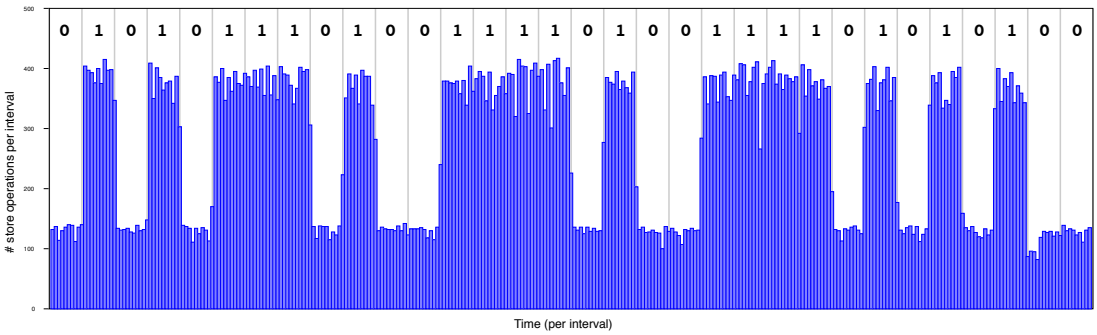

Figure 5: Trace of the disk contention covert channel, showing the operations per interval for the receiver thread.

0's. Next, the sender iterates over this sequence, and when it encounters a 0 bit, it will try to saturate the shared resource during a predefined time interval. As a result, the receiver will observe a performance degradation during this time interval, indicating that a 0 bit was transmitted. When the sender encounters a 1 bit, it will simply remain inactive during the time interval. This allows the receiver to obtain exclusive - aside from other applications - use over the shared resource, resulting in a high throughput. Once the complete message has been sent, the receiver will have obtained a series of performance measurements, which it then has to decode into the binary stream. This can be done by splitting the series into discrete intervals and decoding them as 1 when a high throughput was observed during the interval, and 0 in case of a noticeable performance degradation.

6.1 Disk Contention

In order to ascertain the presence of covert channels in modern web browsers, and assess their practicability, we created an implementation that leverages the disk as shared resource. The reason why the disk was selected is fourfold. First, the bandwidth of hard disks is relatively stable and limited to a few hundred MB/s; this allows the browser process to saturate the entire bandwidth without causing an other resource to become the bottleneck. Second, the use of the hard disk in general is relatively small; while browsing, the disk usage is typically limited to caching resources, which is bound by the network speed, and thus only consumes a fraction of the total bandwidth. Third, because the overall use of the hard disk is limited, the saturation of this resource will not disrupt normal browsing behavior; during our experiments we did not encounter any observable performance degradation, indicating that this covert channel can be used without alerting the victim. Furthermore, we evaluated our experiments under various conditions, ranging from restricting background noise, to a more realistic setting: opening 10 tabs of well-known news websites and social networks, including tabs streaming videos. We did not encounter any significant performance degradation due to the presence of the other tabs. Last, modern browsers provide various APIs that can be used to write large blobs to the disk and later retrieve them; by writing large blobs, the disk operation will be the primary operation, reducing the impact of other resources.

In our implementation of the covert channel, we used the IndexedDB API, which is designed as a low-level API that provides a transactional database system that can be used to store large amounts of data [29]. Of course, other APIs can be used as well, e.g. the Cache API, as long as they provide functionality that can be abused to saturate the hard disk's bandwidth. In our receiver session, we constantly update the value of a single key to a precomputed random string of 512kB, using the $\mathrm{put}()$ operation. We then measure the number of update operations that can be performed within a single time interval, which for the purpose of this proof-of-concept, we set to 200 milliseconds. In the sender session, we use the same update operations to communicate the 0 bit values. Here, we used an interval of 2 seconds, allowing the receiver to obtain 10 measurements per interval.

Figure 5 shows a trace of the performance measurements observed by the receiver (an incognito tab in Google Chrome). This trace clearly shows that during the intervals the sender (a regular tab in Google Chrome) was active, the receiver's observed performance is reduced by more than half. Within little over a minute, the sender managed to communicate a sequence of 32 bits (the ASCII encoding of the string "WOOT"[11]). The time required to perform the attack can be significantly reduced by minimizing the time interval as well as by using a ternary or quaternary encoding instead of a binary one. In case of a ternary encoding, the adversary will use 2 concurrent sender threads instead of a single one (2 concurrent threads will further reduce the performance of the receiver thread, thus providing 3 levels of observed performance).

[11] W: 01010111 O: 01001111 O: 01001111 T: 01010100

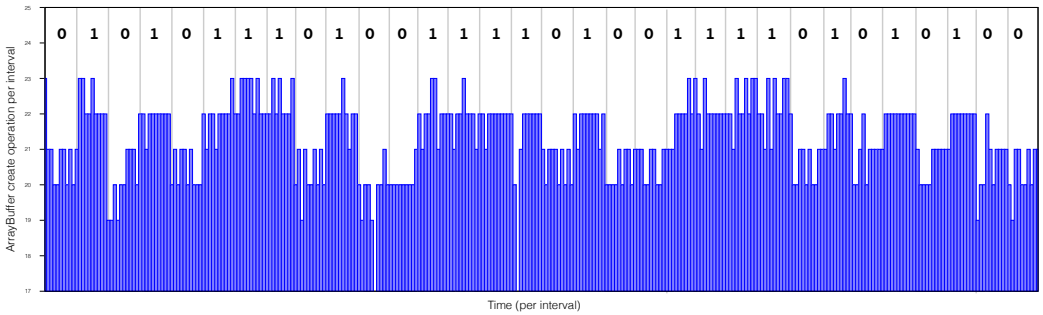

Figure 6: Trace of the ArrayBuffer*-based covert channel, showing the number of times a buffer could be created per interval.*

6.2 Memory Contention

It should be noted that Tor Browser, and Firefox in private browsing mode, disable IndexedDB, and, to the best of our knowledge, all other operations that write to the disk. As such, these aforementioned covert channel cannot be applied here. Nevertheless, we found that other resources can be used to set up covert channels as well. More precisely, we found that when two browser sessions try to concurrently allocate memory, a minor but still measurable performance degradation occurs. To evaluate this memory-based covert channel, we set up a similar experiment as with the covert channel based on disk contention. The memory contention was exploited by iteratively creating ArrayBuffer elements of 20MB in size; this triggers the browser to allocate memory for the buffer [28]. For this experiment, the sender was a tab in Tor Browser, and the receiver a tab in a regular Chrome browsing session. Additionally, because of the reduced timer resolution in Tor Browser, the sending interval was set to 4 seconds, and the receiving interval was changed accordingly, to 400ms.

The trace obtained by the receiver is shown in Figure 6. Compared to the disk-based covert channel, the performance degradation caused by the resource contention is less pronounced. Nevertheless, by computing the median value for each interval, these are all consistently smaller for 0 bits (values of 21 or lower) than for 1 bits (values of 22 or higher). This shows that the attack still remains feasible. It should be noted that because the timer in Tor Browser introduces random noise, the intervals of send and receiver slightly drifted. As such, the grid lines indicating the intervals were slightly tweaked to account for this. In a real-world setting, an adversary could create a more accurate timer, following the techniques introduced by Schwarz et al. [37], or use a more intelligent way to automatically decode the measurements back into a binary sequence. The consequences of a cross-browser covert channel affecting Tor Browser can be considered more severe, as it can be used to reveal the victim's IP address.

7 Discussion

7.1 Attack Consequences

In Section 4, we report on a newly introduced timing attack that can be used to detect the presence of browser extensions in Google Chrome. Because the attack can be launched, even when the user is in incognito mode, an adversary could enumerate the extensions that have been installed and use that as a fingerprint. Interestingly, because Chrome extensions are by default synchronized across all browsers the user logs in to, such a fingerprint even allows tracking a user across devices, assuming they are logged in to the same account on their browser.

In Section 5, we evaluate the potentially nefarious consequences of events that fire across browsing sessions. We find that in their current implementation, browsers will simultaneously fire events that are caused by an external event, e.g. when the client disconnects or reconnects to the network. Although these events may only occur sporadically, depending on the user's environment, a single event suffices to link together all concurrently opened browser sessions. More precisely, when a user has a Chrome and Firefox browser open (possibly in the background), and a piece of JavaScript code is executed on at least one page per browser (Englehardt et al. show that content of certain third-parties is included by more than 50% of the websites), the adversary can link together the two, otherwise completely separated, browser sessions.

In Section 6, we show that covert channels that are caused by resource contention can be an effective way of covertly communicating identifiers across different browsers. We report on a covert channel that leverages disk contention, as well as one that exploits contention at the memory level. Our evaluation shows that the disk-based attack is highly effective and robust against background noise. This attack can be used to associate sessions of concurrently opened browsers, or sessions in regular mode to incognito mode, with the exception of Tor Browser and Firefox's Private Browsing mode, as

these prevent writing to the disk. Our follow-up attack, which exploits memory contention, serves as a strong indicator that resource-contention attacks are not limited to a single type of resource, with varying levels of success. With this attack, we show that even browsers that provide the highest level of anonymity can be subjected to resource contention attacks. Our attack proved successful against Tor Browser, which means that an adversary can exploit it to reveal the IP address of a user in case the user has a concurrent browsing session that is not over Tor. Due to the network overhead imposed by the anonymity network, and because various websites block all traffic originating from Tor, we consider it probable that users simultaneously use a different browser for visiting websites that are not be considered sensitive.

7.2 Defenses

Because the session-associating techniques span a variety of classes, and stem from various side-channels, we consider it highly unlikely that a single countermeasure would be able to defend against all attacks. Nevertheless, there are a number of countermeasures that could be applied, which would make exploiting the side-channels more difficult. For the timing attack against browser extensions, we suggest making the extension URLs unguessable, as is the case with Firefox, or removing the timing side-channel from the implementation. This is discussed in more detail in Section 4.2.2.

To counter the attack that leverages cross-session events, we suggest to prevent firing the same event at exactly the same time across different tabs. For instance, when events are only fired on active tabs, this significantly reduces the attack surface. Although an adversary could still try to bypass this, for instance running his code in a pop-under window, which may go unnoticed for an extended time period, this defense makes a real-world attack significantly more difficult to successfully execute.

For the covert channels that abuse resource contention, defenses are not straightforward. One of the key goals of modern browsers is to load web pages as fast as possible. To achieve this goal, browsers try to utilize system resources to their fullest extent, which is in fact the main cause of the resource contention side-channel. A potential defense could be to restrict the resource usage for every tab (including the background processes they launch, e.g. as service workers or web workers). However, this could cause a significant performance overhead when applied naively. Nevertheless, when these restrictions are only imposed on inactive tabs[12] (along with their background threads), the performance overhead would become less noticeable, and similar to the defense against cross-session events, it limit the adversary's ability to successfully execute the attack in a real-world setting.

In our evaluation, we only considered disk-based and memory-based contention side-channels. We consider it an interesting avenue for future work to further explore which other resources could be leveraged to create a covert channel, as well as evaluate their impact on possible defenses.

8 Related Work

Browser Fingerprinting

Prior work in the area of fingerprinting and online tracking has primarily focused on creating unique fingerprints by extracting information at various levels. At the level of the browser, Eckersley found that by enumerating various properties, such as version and configuration information, online users can be uniquely identified [9]. More recently, it was found that the presence of browser extensions can be detected, either by observing changes they introduce to the DOM [39], or by levering resources from the $\mathrm{web\_accessible\_resources}$ property [38]. In contrast to the previously known techniques that detect the presence of browser extensions, the timing attack described in Section 4 can, at the time of writing, be applied even when the extension is inactive, e.g. in incognito mode.

At the level of the system, prior work has focused on fingerprinting by means of enumerating system configurations [4, 6] and installed fonts [9, 33, 2]. Since the enumeration of these system properties will be identical across browsers, these fingerprints can also be used to associate isolated browsing sessions, provided the fingerprint is sufficiently unique. Similarly, it has been found that a unique fingerprint can be extracted from various hardware components, For instance, the way the GPU renders drawings of a $<\mathrm{canvas}>$ element can be used to create a unique fingerprint [27, 1, 6]. Other hardware components such as the accelerometer and gyroscope have also been shown to leak fingerprintable information [7, 5].

Side-channel Attacks

The tracking techniques introduced in this paper exploit side-channels, which, to some extent, resemble side-channel attack that have been performed in different contexts. For instance, in Section 4, we exploit a timing attack by measuring the time required to perform a fetch operation. Such timing attacks have been known for

[12] Since Google Chrome version 57, the CPU usage of background tabs is restricted, however this does not apply to tabs playing audio [15]. As such, an adversary could play a silent audio file in the offending tab. Moreover, since only CPU wall time is affected, it is uncertain whether this would counter all covert channels based on resource contention.

several decades, and in the context of the web, have been introduced by Felten et al. [13]. Furthermore, Van Goethem et al. have shown that timing side-channels exposed by browsers can be used to leak the size of cross-origin resources [40]. The timing attack introduced in this paper is similar, in the sense that both attacks measure the time required for the browser to perform an action, thereby leaking sensitive information. More recently, Vela and Köpf discovered a timing attack against shared event loops in Google Chrome which can be exploited to create a covert channel or detect which websites a user is visiting.

The resource contention covert channel presented in Section 6, was also exploited by Ristenpart et al. in the context of cloud computing [35]. In their attack, they used the covert channel to determine co-residency of two EC2 instances.

9 Conclusion

A wide variety of methods have been proposed and implemented to thwart online tracking and fingerprinting. In this paper, we show that existing solutions, which typically apply defenses retroactively, fall short of protecting users against attacks that link together separate browsing sessions. As a result of assessing various components of the browser, guided by the interactions it makes with its surrounding environment, we discovered 3 classes of side-channels that are highly effective at associating browsing sessions, even across browsers. We show that these side-channels, which are based on timing, making simultaneous observations, and resource contention, are able to circumvent existing privacy-enhancing solutions. When current threat models on fingerprinting and online tracking are extended with our newly introduced attacks, browser vendors and specification developers have a more accurate view on the threat surface. This allows them to discover similar issues more easily and work towards privacy-preserving solutions.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This research is partially funded by the Research Fund KU Leuven.

References

[1] ACAR, G., EUBANK, C., ENGLEHARDT, S., JUAREZ, M., NARAYANAN, A., AND DIAZ, C. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 674–689.

[2] ACAR, G., JUAREZ, M., NIKIFORAKIS, N., DIAZ, C., GÜRSES, S., PIESSENS, F., AND PRENEEL, B. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 1129–1140.

[3] BODA, K. Firegloves.

[4] BODA, K., FÖLDES, Á. M., GULYÁS, G. G., AND IMRE, S. User tracking on the web via cross-browser fingerprinting. In *Nordic Conference on Secure IT Systems* (2011), Springer, pp. 31–46.

[5] BOJINOV, H., MICHALEVSKY, Y., NAKIBLY, G., AND BONEH, D. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416* (2014).

[6] CAO, Y., LI, S., AND WIJMANS, E. (cross-)browser fingerprinting via os and hardware level features. In *NDSS* (2017).

[7] DEY, S., ROY, N., XU, W., CHOUDHURY, R. R., AND NELAKUDITI, S. Accelprint: Imperfections of accelerometers make smartphones trackable. In *NDSS* (2014).

[8] DOTY, N. Re: Comments/Questions on Media Capture Streams âĂŞ Privacy and Security Considerations. https://lists.w3.org/Archives/Public/public-privacy/2015OctDec/0028.html, 2015.

[9] ECKERSLEY, P. How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium* (2010), Springer, pp. 1–18.

[10] ELECTRONIC FRONTIER FOUNDATION. Privacy Badger. https://www.eff.org/privacybadger, 2017.

[11] EMARKETER. US Ad Spending: The eMarketer Forecast for 2017. https://www.emarketer.com/Report/US-Ad-Spending-eMarketer-Forecast-2017/2001998, March 2017.

[12] 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* (2016), ACM, pp. 1388–1401.

[13] FELTEN, E. W., AND SCHNEIDER, M. A. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and communications security* (2000), ACM, pp. 25–32.

[14] GHOSTERY. Cleaner, faster, and safer browsing. https://www.ghostery.com/, 2017.

[15] GOOGLE CHROME. Background Tabs in Chrome 57. https://developers.google.com/web/updates/2017/03/background_tabs, 2017.

[16] GOOGLE CHROME. Manifest - Web Accessible Resources. https://developer.chrome.com/extensions/manifest/web_accessible_resources, 2017.

[17] KETTLE, J. Sparse bruteforce addon detection. http://www.skeletonscribe.net/2011/07/sparse-bruteforce-addon-scanner.html, 2011.

[18] KOTOWICZ, K. Intro to Chrome addons hacking: fingerprinting. http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html, 2012.

[19] KRISHNAMURTHY, B., AND WILLS, C. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web* (2009), ACM, pp. 541–550.

[20] LIBERT, T. Exposing the invisible web: An analysis of third-party HTTP requests on 1 million websites. *International Journal of Communication 9* (2015), 18.

[21] MARIÉ, D. Chrome Extensions Archive. https://crx.dam.io/, 2017.

[22] MAYER, J. R. Any person... a pamphleteerâĂİ: Internet anonymity in the age of web 2.0. *Undergraduate Senior Thesis, Princeton University* (2009).

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

[24] MCDONALD, A. M., AND CRANOR, L. F. A survey of the use of adobe flash local shared objects to respawn http cookies. *ISJLP 7* (2011), 639.

[25] MIAGKOV, A. Chameleon. https://github.com/ghostwords/chameleon, 2017.

[26] MOWERY, K., BOGENREIF, D., YILEK, S., AND SHACHAM, H. Fingerprinting information in javascript implementations. *Proceedings of W2SP 2* (2011), 180–193.

[27] MOWERY, K., AND SHACHAM, H. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP* (2012), 1–12.

[28] MOZILLA DEVELOPER NETWORK. ArrayBuffer. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer, 2017.

[29] MOZILLA DEVELOPER NETWORK. IndexedDB API. https://developer.mozilla.org/en/docs/Web/API/IndexedDB_API, 2017.

[30] MOZILLA DEVELOPER NETWORK (MDN). web_accessible_resources. https://developer.mozilla.org/en-US/Add-ons/WebExtensions/manifest.json/web_accessible_resources, 2017.

[31] MULAZZANI, M., RESCHL, P., HUBER, M., LEITHNER, M., SCHRITTWIESER, S., WEIPPL, E., AND WIEN, F. Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)* (2013), vol. 5.

[32] NIKIFORAKIS, N., JOOSEN, W., AND LIVSHITS, B. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web* (2015), ACM, pp. 820–830.

[33] NIKIFORAKIS, N., KAPRAVELOS, A., JOOSEN, W., KRUEGEL, C., PIESSENS, F., AND VIGNA, G. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and privacy (SP), 2013 IEEE symposium on* (2013), IEEE, pp. 541–555.

[34] PERRY, M., CLARK, E., MURDOCH, S., AND KOPPEN, G. The design and implementation of the Tor Browser. https://www.torproject.org/projects/torbrowser/design/, 2017.

[35] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 199–212.

[36] 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* (2012), USENIX Association, pp. 12–12.

[37] SCHWARZ, M., MAURICE, C., GRUSS, D., AND MANGARD, S. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript.

[38] SJÖSTEN, A., VAN ACKER, S., AND SABELFELD, A. Discovering browser extensions via web accessible resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (2017), ACM, pp. 329–336.

[39] STAROV, O., AND NIKIFORAKIS, N. Xhound: Quantifying the fingerprintability of browser extensions.

[40] VAN GOETHEM, T., JOOSEN, W., AND NIKIFORAKIS, N. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1382–1393.

[41] WILLIAMS, B. Adblock Plus and (a little) more. https://adblockplus.org/blog/decadblock-adblock-plus-turns-10, 2016.