Libor Polčák*, Marek Saloň, Giorgio Maone, Radek Hranický, and Michael McMahon

JShelter: Give Me My Browser Back

Abstract: The Web is used daily by billions. Even so, users are not protected from many threats by default. This position paper builds on previous web privacy and security research and introduces JShelter, a webextension that fights to return the browser to users. Moreover, we introduce a library helping with common webextension development tasks and fixing loopholes misused by previous research. JShelter focuses on fingerprinting prevention, limitations of rich web APIs, prevention of attacks connected to timing, and learning information about the computer, the browser, the user, and surrounding physical environment and location. We discovered a loophole in the sensor timestamps that lets any page observe the device boot time if sensor APIs are enabled in Chromium-based browsers. JShelter provides a fingerprinting report and other feedback that can be used by future security research and data protection authorities. Thousands of users around the world use the webextension every day.

Keywords: Browser fingerprinting, web privacy, web security, webextension APIs, JavaScript

1 Introduction

Most people interact with web pages daily. Nowadays, many activities are often carried out exclusively in a Web browser, including shopping, searching for travel information, performing leisure activities such as gaming, business and office work. For several years, browser vendors have been adding new JavaScript APIs to solicit the development of rich web applications [60].

Web visitors are subject to hostile tracking [10, 13, 24, 36, 48], fingerprinting [26, 32], and malware [6, 46]. Some of the recently added APIs influence the privacy of the users. For example, the Geolocation API[1] is beneficial for navigation in the real world. However, users might not be willing to share the location with all visited sites. In the case of Geolocation API, browsers ask users for permission but not all APIs need user permission. Users cannot limit the precision of the Geolocation API. However, sometimes they want to share a more precise location (e.g. during navigation), and other times they want to share the location with limited precision (e.g. they are exploring a location unrelated to their current position).

This paper presents JShelter, a web browser extension (webextension) that allows users to tweak the browser APIs. Additionally, JShelter detects and prevents fingerprinting. Moreover, JShelter blocks attempts to misuse the browser as a proxy to access the local network. JShelter educates users by explaining fingerprinting APIs in a report. JShelter integrates several previous research projects like Chrome Zero [39] and little-lies-based fingerprinting prevention [44, 50]. We needed to solve reliable injection of modifications to all visited pages, iframes, and web workers. We introduce NoScript Commons Library (NSCL)[2] that other privacy- and security-related webextensions can reuse to solve common tasks. We implemented JShelter for Firefox and Chromium-based browsers like Chrome, Opera, and Edge. We provide experience from user feedback that should be valuable to other research projects.

The evaluation shows that JShelter prevents many attacks, including learning (1) browser and computer fingerprints, (2) user biometrics, (3) computer clock-skew, and (4) running applications. Sensors available for all pages on Android make user open to several attacks [5, 22, 67]. JShelter prevents the danger by pretending to be a stationary device. JShelter mitigates

***Corresponding Author: Libor Polčák:** Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic, E-mail: polcak@fit.vut.cz
Marek Saloň: Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic
Giorgio Maone: Hackademix, via Mario Rapisardi 53, 90144 Palermo, Italy, E-mail: giorgio@maone.net
Radek Hranický: Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, E-mail: ihranicky@fit.vut.cz
Michael McMahon: Web developer, Free Software Foundation, 51 Franklin Street, Fifth Floor, Boston, MA 02110 USA, E-mail: michael@fsf.org

1 https://developer.mozilla.org/en-US/docs/Web/API/Geolocation
2 https://noscript.net/commons-library

leaking boot time of the device through sensor timestamps in Chromium-based browsers.

This paper is organised as follows. Section 2 presents the threats that users face while web browsing. Section 3 compares JShelter to other security- and privacy-related webextensions. Section 4 provides the design decisions that we faced during the development of JShelter. Section 5 evaluates the JShelter features and discusses user feedback. Section 6 concludes this paper.

2 Threats

This section presents threats that every web user faces every time they load and execute unknown JavaScript code. Although modern browsers employ security measures, such as same-origin policy[3], there are still threats that are not mitigated.

2.1 T1: Detail user behaviour monitoring

In theory, laws like GDPR and ePrivacy Regulation give each person control over their personal data and devices. However, there is a significant lack of control over personal data on the web [8, 24, 36, 37, 52] in practice. The advertisement technologies are under considerable scrutiny in Europe [1, 10], but tracking scripts are omnipresent on the web. Users risk complete disclosure of their browsing history.

Web-content providers want detailed information on user interaction with their web pages. JavaScript-event listeners and handlers can track user activities such as mouse movement, typing and clicking [13]. The website operator can replay the user session in real-time or later. Customer services provide chat windows providing information in real-time. However, some libraries for chat interaction transfer the question while the user types instead of waiting for the user to press the send button [23].

Previous literature focused on processing behavioural biometric features derived from input user interaction (keyboard, mouse, and touch events). For example, it is possible to uniquely identify users [28, 70], derive handedness [47], or age and gender [48].

2.2 T2: Browser and computer fingerprinting

Historically, trackers stored user identifiers in third party cookies during T1 tracking. However, browser vendors limit third party cookies. Hence trackers move to alternative ways of identifying users. Browser and computer fingerprinting is a stateless tracking method that tries to find features that make (almost) every browser uniquely identifiable [9, 12, 32, 33]. For example, the content of HTTP headers, including user agent string, screen size, language, time zone, and system fonts, together with hardware-dependent characteristics such as canvas image rendering [9, 40], audio processing [14], installed fonts [16], installed browser extensions [21, 54, 55, 62], the sites that the user is currently logged in [21], clock skew [30, 51] and other techniques [32]. The goal of the fingerprinter is to create a stable identifier of a user so that the user is identifiable on different sites. Computer fingerprint is the same in every browser on the same computer, while browser fingerprint differs for different browsers running on the same computer. Recent studies have shown that user tracking is becoming more prevalent and complex [35]. Note that the leaking information may uncover vulnerabilities of the fingerprinted systems, and a fingerprinting database can be a valuable source of information for an adversary wanting to misuse the data.

A fingerprint is considered passive when it contains natively accessible information from HTTP headers or network traffic. On the other hand, active fingerprint runs JavaScript code to retrieve data from browser APIs. One of the goals of JShelter is to prevent active fingerprinting.

Several studies monitored the deployed fingerprinting techniques on the Internet [3, 4, 14, 17, 26, 45]. Trackers exploit evercookies, shared cookies, font enumeration, canvas, web audio, WebRTC, and many other APIs to identify browsers and consequently their users. However, studies found current countermeasures insufficient for a dedicated fingerprinter willing to reveal inconsistencies in API readings [54, 68].

Transparency and Consent Framework (TCF) was supposed to make online advertisement compliant with ePrivacy and GDPR. However, the Belgian data protection authority recently confirmed many flaws [10]. Version 2 of the framework allows companies to self-report active and passive fingerprinting. Figure 1 depicts

3 https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

publicly available data by IAB Europe[4] and shows that more than 400 companies passively fingerprint users and more than 100 companies actively use JavaScript APIs to create a unique fingerprint.

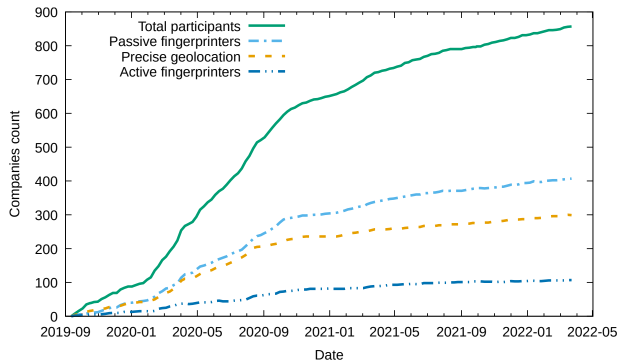


Fig. 1. TCF participants reporting fingerprinting activities and precise geolocation processing.

Current research distinguishes targeted and not targeted fingerprinting [32]. Not targeted fingerprinting focuses on observing visiting browsers or computer fingerprints and trying to link their identity to a previous visitor. Targeted fingerprinting tries to detect a tailored fingerprint of an individual, for example, for law enforcement investigations [54].

Browser fingerprinting can also be used for benign use cases like multifactor authentication — if a website detects that a user connects from the same device as previously seen, it is not necessary to perform additional authentication steps. A website can recommend installing critical security updates based on system properties, like the browser version. Some websites collect browser fingerprints to distinguish between human users and bots to prevent fraud.

2.3 T3: Very rich browser APIs

Modern websites offer the capabilities of native applications. Browsers support video calls, audio and video editing, maps and navigation, augmented and virtual reality. One can control games with gamepads or check the battery status. The webpage may change its appearance according to the ambient light. Nevertheless, most web pages do not need these advanced APIs [61]. Some APIs like Geolocation or microphone and camera access need explicit approval by users. Others like gamepads, virtual reality, battery, or sensors are available for all visited pages[5]

Iqbal et al. [26] detected misusing the APIs by many fingerprinting scripts. Both Generic Sensor W3C Candidate Recommendation Draft[6] and literature mention several risks stemming from sensor reading like location tracking [22], eavesdropping, keystroke monitoring, device fingerprinting [67], and user identification [5]. Fig. 1 reports TCF data on companies using precise geolocation data (precision better than 500 meters).

2.4 T4: Hostile third-party scripts

Whenever a user visits a web page, it can include external scripts using the script element (for example, the script provided by an advertisement provider or the script performing visitor analysis, see T1). All scripts, including the external scripts, can access the document object model (DOM) of the web page and have the same capabilities as scripts hosted on the same domain. Consequently, if a first-party can access sensor data (T3), a third-party script can access the same data. Note that trackers (T1) are typically third-party scripts.

DOM dynamically reflects changes on the page, including password and credit card strings. Several researchers [2, 58, 64] warn that malicious scripts can leverage the autofill functionality of password managers to leak user credentials without their awareness. Other research focused on contact forms that leak personal data to unintended recipients [63]. Some packages aim at stealing money [53].

4 See https://vendor-list.consensu.org/v2/archives/vendor-list-vNUM.json where NUM is the number of the week since the start of the framework. See https://www.fit.vutbr.cz/~polcak/tcf/tcf2.html for more data from the framework.

5 Sensor APIs are currently implemented, or partially implemented, in Chromium-based browsers like Chrome, Edge, and Opera. For Android devices, the support exists in Chrome for Android, Opera for Android, and various Chromium-based browsers like Samsung Mobile or Kiwi Browser. The concrete support for individual classes depends on the browser type and version. Some features are considered experimental and only work when browser flags like `#enable-experimental-web-platform-features` or `#enable-generic-sensor-extra-classes` are enabled. Sensor APIs are enabled by default in Chrome on Android.

6 https://www.w3.org/TR/2021/CRD-generic-sensor-20210729/#main-privacy-security-threats

Modern web development includes many libraries. Decan et al. [11] studied the npm ecosystem — package vulnerabilities and the time needed to fix a vulnerability. They observed that it often takes a long time to discover vulnerabilities. It is widespread for websites to include libraries with known security vulnerabilities [34]. Lauinger et al. [34] observed that websites use unpatched libraries for years. Additionally, they observed that libraries included transitively or via advertisement trackers are more likely to be vulnerable as the ecosystem is complex, unorganised, and it is often hard to identify the vulnerable package versions. Sometimes, one web page includes multiple versions of the same library simultaneously. Mush et al. [42] explored that 25 % of all sites affected by client-side cross-site scripting are only vulnerable due to a flaw in the included third party code.

Figure 2 shows the number of pages with detected vulnerabilities created by HTTP Archive[7]. The pages are becoming less vulnerable in the last years, but 58.9 % of pages are vulnerable to at least one known and detected vulnerability.

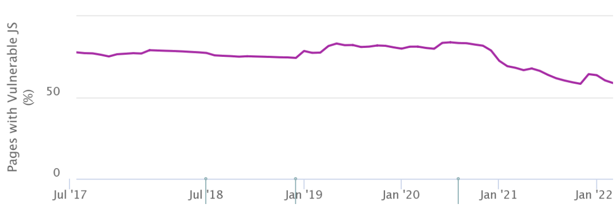


Fig. 2. Web pages with detected vulnerabilities by HTTP archive.

2.5 T5: Local network scanning

Devices browsing the web are typically connected behind NAT, which does not allow external hosts to open connections to devices in the local network (e.g. printers). Although the same-origin policy does not allow a web page to access arbitrary resources, there are side channels that might provide enough information about an existence of a resource, including resources in the local network [6]. A web page can try to exploit the web browser as a proxy between the remote website and resources in the local network. Bergbom [6] demonstrated that it is possible to execute arbitrary commands on a local machine under certain circumstances (in this case, it was an insecure Jenkins configuration).

2.6 T6: Microarchitectural attacks

Previous research also focused on side-channel attacks that can reveal what the user has recently done with the computer. For example, content-based page deduplication performed by an operating system or a virtual machine hypervisor can reveal if specific images or websites are currently opened [19] on the same computer (hardware), possibly on another virtual machine. The reply time for a specific request depends on the cached content, so the reply time reveals if the content was recently visited [15]. Moreover, even uncached content leaks information on the server state [7]. Bortz and Boneh [7] studied server reply times influenced by different code paths taken by the server and were able to reveal private information. The `requestAnimationFrame` API can be used to time browser rendering operations and reveal information on browser history and read pixels from cross-origin iframes [65].

Operating systems isolate processes from each other and the kernel. However, deficiencies in hardware can provide possibilities to circumvent the isolation. Gruss et al. [20] exploited JavaScript to modify memory cells belonging to different processes (the attack is called Rowhammer). Hence, they gained unrestricted access to systems of website visitors. They exploited operating systems optimisations and high-precision timings [20]. Later, Gruss et al. [18] showed that industry countermeasures against Rowhammer attacks are ineffective. Spectre attacks can be executed from JavaScript and leak data in the memory of other processes running on the same system [29].

Some websites provide different content based on age, gender and location. Van Goethem et al. [66] employed timing attacks to reveal data about users by measuring the size of the reply for resources with different contents for different users.

Smith et al. [59] determined browser history by the visited link pseudoclass and timing redrawing of the links based on the target URL.

7 https://httparchive.org/reports/state-of-the-web#pctVuln

3 Countermeasures

Many popular security and privacy-enhancing approaches already exist. Let us focus on existing tools addressing the threats raised in §2.

3.1 Browser extensions

Adblockers and other tracker blockers typically address threats T1 and T2 but can also address T5 and T6. The blockers employ lists of URLs or parts of URLs that are considered harmful to user privacy or security. The advantage for the user is that many tools focus on blocking (for example, uBlock Origin, EFF Privacy Badger, Ghostery) and also blocklists that are usually compatible with several blockers. Browsers like Firefox [31] and Brave include tracking prevention by default. The downside is that it is easy to evade blockers [38]. The malicious web server needs to change the name of the script. For example, one of the Czech banks is currently being investigated for including tracker scripts in their internet banking. The default uBlock Origin blocklists did not match the trackers. Hence, blocklists are very useful as a first-line defence and improve web performance [31]. However, blockers are not enough as the niche cases evade the blockers [38].

Webextensions like NoScript Security Suite and uMatrix Origin allow users to block JavaScript or other content either completely or per domain. Hence, they can address all six threats raised in §2. However, the user needs to evaluate what scripts to allow. HTTP Archive reports[8] that an average page includes 22 external requests (21 requests for mobile devices). Many pages depend on JavaScript. Users must select what content to trust. A typical page contains resources from many external sources, so such a user requires excellent knowledge. Moreover, a malicious code may be only a part of resources; the rest of the resource can be necessary for correct page functionality. So we believe that webextensions like NoScript Security Suite and uMatrix Origin are good but do not protect the user from accidentally allowing malicious code.

JavaScript Zero [39] (also known as Chrome Zero[9]) expects that a user lets the browser run the vulnerable code and focuses on mitigating T6. Even most skilful users can run malicious code if the script URL evades blocklists and other parts of the code are needed for the page to display correctly. However, the practical implementation supports only Chromium-based browsers, is not maintained since 2017, and Shusterman et al. [57] have shown that the webpage can obtain access to the original API calls.

Web API Manager [61] classifies JavaScript APIs into 81 standards[10]. A the Web API Manager webextension user can disable all functionality defined by any of the standards. The authors prepared three configurations with standards blocked depending on their benefits and costs [61]. Web API Manager is most effective against T3 and not targeted T2, but it can help mitigate other threats. Unfortunately, Web API Manager does not allow a user to allow only a part of the standard, e.g. it is not possible to allow Canvas API for drawing but disallow reading that is used for fingerprinting [40]. Additionally, the webextension is no longer maintained[11], it is not compatible with Firefox Multi-Account Containers[12], and it suffers from the Firefox bug related to Content Security Policy (CSP) [41]. A Web API Manager user with a tailored configuration can potentially be uniquely identified with the JavaScript enumerating code developed by Schwarz et al. [54].

We suggest installing cookie managers (threat T1) and local CDN cachers (threats T1 and T4) as other complementary webextensions.

3.2 Privacy-focused browsers

Tor is a network of onion routers that allow relaying TCP connections so that the server does not learn the IP address of a client but an IP address of their Tor exit node. Torbrowser is a Firefox fork that tries to make every instance as uniform as possible. For example, every user should browse with the same window size. However, a fingerprinter can still learn some information like the underlying operating system [32]. Torbrowser also disables several APIs like WebGL. Consequently, Torbrowser is a very good solution to tackle threats T1, T2, T3, T5, and T6.

8 https://httparchive.org/reports/page-weight?start=earliest&end=latest&view=list#reqJs

9 https://github.com/IAIK/ChromeZero

10 https://github.com/snyderp/web-api-manager/tree/master/sources/standards

11 See the message on the GitHub page https://github.com/snyderp/web-api-manager/blob/master/README.md

12 See https://github.com/snyderp/web-api-manager/issues/53 for more details

Nevertheless, Torbrowser users should not resize the window and install additional webextensions. These requirements downgrade comfort, and users might be unwilling to abandon favourite webextensions or be tempted to resize the window for more comfort. As the communication is relayed multiple times by relays spread worldwide, latency increases, and throughput is limited. The list of Tor exit node IP addresses is public. Moreover, malicious actors often misuse Tor. Some services block Tor traffic, either to prevent frequent attacks or as a temporary measure to block an attack.

Brave browser is a Chromium fork that focuses on privacy. For example, it has a built-in blocker and anti-fingerprinting solution. Using Brave is a good option to tackle T1, T2, T5, and T6. A disadvantage is the long build time. Often, it is not available in GNU/Linux distribution repositories.

Mozilla is working on integrating fingerprinting resisting techniques from Torbrowser[13] to Firefox (Firefox Fingerprinting Protection, also known as resist fingerprinting). However, the work is not done, and it is a possible related research question if the hiding in the herd strategy makes sense before it is adopted for all users. Moreover, inconsistencies arise. For example, Torbrowser does not implement WebGL. As Firefox adopts fingerprinting protections from Torbrowser, Firefox modifies readings from 2D canvas and does not modify WebGL canvas. That creates a false sense of protection.

3.3 Current browser fingerprinting countermeasures

Let us summarise the anti-fingerprinting protections of the tools covered in this section. Modifying the content of fingerprints is a valid choice to resist a fingerprinting attempt. However, each modification may create an inconsistency that may improve the fingerprintability of the browser [32]. Currently, three anti-fingerprinting approaches exist.

(1) *Create homogeneous fingerprints.* If the commonly used fingerprinting APIs returned the same values in every browser, a fingerprinter would not be able to construct a fingerprint and tell the users behind the browsers apart. The leading representative of this approach is Torbrowser. Unfortunately, homogeneous fingerprints have an inherent downside of following specific rules to be effective. Most importantly, the effectiveness of the approach depends on the broad coverage of the blocked APIs and the size of the population employing the countermeasures. All browsers with the same fingerprint form an anonymity set [49]. An observer cannot distinguish between browsers in the anonymity set. With every missed fingerprintable attribute, the anonymity set breaks into smaller sets. For example, Torbrowser strongly recommends using a specific window size. Suppose a user changes a window size to a value different from all other Torbrowser users. In that case, a fingerprinter can identify the user solely by this attribute. Moreover, Torbrowser hides the IP address of the user. A webextension cannot hide or mask the IP address.

(2) *Change the fingerprints on different domains to disable cross-domain linkage.* Brave browser also modifies the results of APIs commonly used for fingerprinting. Its goal is to create a unique fingerprint for each domain and session. As the output of APIs commonly used for fingerprinting changes for every visited domain, it cannot be used for cross-domain linking of the same browser.

(3) *Detect and block fingerprint attempts.* A protection tool can monitor access to properties commonly misused for fingerprinting and block access to additional properties or limit the page ability to upload the fingerprint. To reliably prevent sharing the fingerprints with trackers, any network traffic to the tracking server has to be blocked and the web page cannot have an opportunity to store the fingerprint for retrieval after page reload. Such measures can be effective against fingerprinting. Nevertheless, they also impose severe restrictions for web applications, limit overall usability, and break page behaviour. Fingerprinting detection can also be imprecise. In practice, it takes time to detect that a fingerprint is indeed being computed. As a page can immediately send the values being read for fingerprinting to the server, the server can learn a partial fingerprint before the fingerprinting is detected and blocked.

4 JShelter design decisions

As the current state-of-the-art covered in §3 suggests, there is no perfect and straightforward solution for the threats raised in §2. This section covers the design decisions of JShelter and the countermeasures we decided to implement.

13 https://bugzilla.mozilla.org/show_bug.cgi?id=1329996

JShelter does not aim at providing a perfect solution either. Our goals are as follows:

1. Create a webextension because webextensions work across multiple browsers and consequently can be easily installed into any browser that supports webextensions, including Firefox and all browsers based on Chromium.
2. Do not create a perfect solution instead focus on what other webextensions lack: a consistent approach to the threat T2 and protection from T3, T4[14], T5, and T6.
3. Make the webextension friendly for people without technical knowledge.

Chrome Zero [39] and Web API Manager [61] were the inspiration for JShelter. Chrome Zero provides examples of protections like closures and Proxy objects. It focuses on microarchitectural attacks. Web API Manager provides a way to selectively disable browser APIs.

Currently, JShelter offers three types of protections. (1) JavaScript Shield (JSS) modifies or disables JavaScript APIs. It aims at threats T2, T3 and time-measurement-related protection for T1 and T6. (2) Fingerprint Detector (FPD) provides heuristic analysis of fingerprinting behaviour and tackles T2, (3) Network Boundary Shield (NBS) monitors the source and destination of each web request and detects attempts to misuse the browser as a proxy to the local network (T5).

4.1 Fingerprint detector

FPD monitors APIs that are commonly used by fingerprinters and applies a heuristic approach to detect fingerprinting behaviour in real-time (see threat T2). When a fingerprinting attempt is detected, FPD notifies the user. The user can configure JShelter to reactively block subsequent asynchronous HTTP requests initiated by the fingerprinting page and clear the storage facilities where the page could have stored a (partial) fingerprint. However, this behaviour may break the page. The goal of the aggressive mode is to prevent the page from uploading the full fingerprint to a server. However, the fingerprinter can gradually upload detected values and a partial fingerprint can leak from the browser.

The heuristic approach was chosen as many prior studies [3, 14, 26, 32] proved it to be a viable approach with a very low false-positive rate. The most challenging part of this approach is a careful selection of detection conditions. The heuristics contain two basic types of entries: (1) JavaScript API endpoints, which are relevant for fingerprinting detection and (2) a hierarchy of groups of related endpoints. For example, we group endpoints according to their semantic properties. Imagine that there are two different endpoints. Both provide hardware information about the device. We can assign both endpoints to a group that covers access to the same hardware properties. The heuristics allow clustering groups to other groups and creating a hierarchy of groups. Ultimately, the heuristics are a tree-like structure that computes the threat that a webpage tried to obtain enough information to compute a unique fingerprint.

FPD is based on previous studies. Iqbal et al. [26] measured the relative prevalence of API keywords in fingerprinting scripts and created a list of APIs using this metric. We extracted selected APIs from the list into groups in our heuristics. We also build upon heuristics proposed by Englehardt and Narayanan [14] to detect additional fingerprinting techniques. We looked through the source code of fingerprinting tools like FingerprintJS[15], Am I Unique[16] and Cover Your Tracks[17]. Furthermore, we analysed existing detection tools, namely A Fingerprinting Monitor For Chrome (FP-MON)[18] and Don't FingerPrint Me (DFPM)[19].

The whole evaluation process dynamically observes the API calls performed by a web page. We analyse the calls themselves. Hence, the dynamic analysis overcomes any obfuscation of fingerprinting scripts.

FPD provides a report that explains why FPD evaluated a visited page as a fingerprinter, see Fig. 3. The report aims to educate users about fingerprinting and report why FPD notified the user and optionally blocked the page. Additionally, the report can be generated from passive observation of web page calls without any JShelter interaction with the page (no API blocking). We expect that researchers will use passive FPD to study fingerprinting in more detail. Additionally, passive FPD

14 Currently, JShelter does not provide any protection for T4 but we evaluate possibilities to add such support in the future. For example, we supervise a diploma thesis in the area that should be defended in 2022.

15 https://github.com/fingerprintjs
16 https://amiunique.org/
17 https://coveryourtracks.eff.org/
18 https://fpmon.github.io/fingerprinting-monitor/
19 https://github.com/freethenation/DFPM

can be employed by European data protection auditors to determine ePrivacy or GDPR breaches.

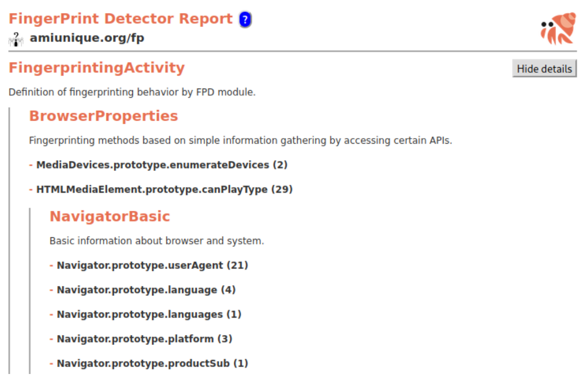


Fig. 3. An excerpt from a FPD report on AmIUnique.org. The user can clearly see what APIs the visited page called.

We expect that the APIs for fingerprinting will change in time so we designed the heuristics to be as flexible as possible. We expect to run periodic web crawls based on the tools initially developed by Snyder et al. [60] and apply machine learning to FPD.

4.2 JavaScript Shield

JSS focuses on spoofing timestamps (threat T1 and T6), fingerprint modifications (threat T2) and limiting APIs available to visited pages (threat T3). JSS offers two predefined profiles that we expect that users should use.

One profile focuses on making the browser appear differently to distinct fingerprinting origins by slightly modifying the results of API calls differently on different domains so that the cross-site fingerprint is not stable [44, 50]. The focus is on applying security countermeasures that are likely not to break web pages.

The other profile focuses on limiting the information provided by the browser by returning fake values from the protected APIs. Some are blocked completely, some provide meaningful but rare values, other return meaningless values. This level makes the user fingerprintable because the results of API calls are generally modified in the same way on all websites and in each session.

JShelter currently modifies 97 APIs inspired by Schwarz et al. [39], Iqbal et al. [26], Snyder et al. [61] and APIs declined by Apple. For each API, we decide its relevance on an individual basis. Usually, we do not modify APIs that are already explicitly permitted by the user, but the analysis might provide an example where the user still wants to limit the precision of the API. For example, Geolocation API allows the page to learn a very precise location while the user might be interested in services in the city. Hence, JShelter allows fine-tuning the precision of the Geolocation.

The slightest mismatch between the results of two APIs can make the user more visible to fingerprinters [32, 43, 54]. Hence, all protections are considered from the point of fingerprintability, the threat to leak information about the browser or user and other threats presented in §2. When it does not require much work, JShelter tries to mimic a stationary device with consistent and plausible readings.

4.2.1 Farbling-like prevention of browser fingerprinting

JSS builds on the Farbling protection implemented in Brave[20] and applies the same or very similar protection. Farbling is, in turn, based on PriVaricator [44] and FPRandom [50]. JSS modifies the values readable by page scripts with small lies that differ per origin. These little lies result in different websites calculating different fingerprints. Moreover, a previously visited website calculates a different fingerprint in a new browsing session. Consequently, cross-site tracking is more complicated.

4.2.2 Interaction between JavaScript Shield and Fingerprint Detector

Both JSS, and FPD aim at preventing fingerprinting. Both are necessary for JShelter.

- The blocking mode of FPD breaks pages. Users are typically tempted to access the content even when they know that they are being fingerprinted. Hence, prevention besides FPD is necessary.
- The JSS profile focusing on limiting information access will likely result in the same fingerprint for all domains; hence, we strongly advise users of this profile to activate FPD.
- We expect most users to stick with the default profile creating little lies. Future research should validate the current approach. For example, JShelter and Brave create indistinguishable changes to canvas readings. These are sufficient for a fingerprinter that creates a hash of the readings. Nevertheless, an

20 See https://github.com/brave/brave-browser/issues/8787 and https://github.com/brave/brave-browser/issues/11770 advanced fingerprinter might, for example, read the colours of specific pixels to determine a presence of a font (different fonts produce a different pixel-wise-long output of the same text). As both Brave and JShelter modify only the least significant bit of each colour, the fingerprinter can ignore this bit and get the information on installed fonts. Hence, we believe that FPD is beneficial as it offers additional protections.

4.2.3 Sensors

JShelter tries to simulate a stationary device and consequently completely spoofs the readings of Geolocation API and AmbientLight, AbsoluteOrientation, RelativeOrientation, Accelerometer, LinearAcceleration, Gravity, Gyroscope, and Magnetometer sensors.

Instead of using the original data, JShelter returns artificially generated values that look like actual sensor readings. Hence the spoofed readings fluctuate around a value that is unique per origin and session. The readings are performed consistently in the same origin tabs, so the same sensor produces the same value in each tab.

We observed sensor readings from several devices to learn the fluctuations of stationary devices in different environments. Most of the sensors have small deviations. However, magnetometer readings have big fluctuations. JShelter simulates Magnetometer fluctuation by using a series of sines for each axis. Each sine has a unique amplitude, phase shift, and period. The number of sines per axis is chosen pseudorandomly. JShelter currently employs 20 to 30 sines for each axis. Nevertheless, the optimal configuration is subject to future research. More sines give less predictable results but also increase the computing complexity, negatively impacting the browser's performance.

The readings of the acceleration and orientation sensors are generated consistently between each other from an initial device orientation that JShelter generates for each origin and session.

4.2.4 User in Control

The number of modified APIs is huge. We expect that users will encounter pages that are broken by JShelter or do not work as expected. For example, the user might want to play games with a gamepad device on some pages or make a call on others.

JSS allows each user to fine-tune the protection for each origin. Based on the reported issues, we decided that some users are not willing to dig into the configuration. Those can disable JSS as a whole with a simple ON/OFF popup switch. More experienced users can react to information provided by FPD and turn off JSS fingerprint protection when the visited site does not behave as a fingerprinter. The most experienced users can fine-tune the behaviour per API group. Figure 4 shows an example of a user accessing a page that allows video calls. The user sees the groups with APIs that have been called by the visited page at the top and can quickly fix a broken page.

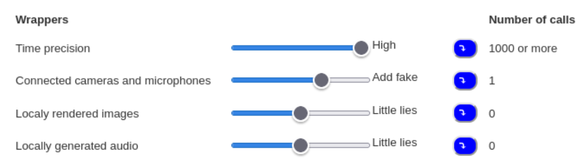


Fig. 4. JSS reports back which APIs are being used by the page.

4.3 Effective modifications of JavaScript environment

Monitoring and modifying the results of the built-in JavaScript APIs and built-in object behaviour is the JShelter core functionality. JShelter employs the same mechanism proposed by Schwarz et al. [39] in Chrome Zero. However, Chrome Zero was a proof-of-concept without any modification in the last four years. Shusterman et al. [57] identified several problems with Chrome Zero:

1. Unprotected prototype chains (issue 1): the original implementation is available through the prototype chain because Chrome Zero protects a wrong property.
2. Delayed JavaScript environment initialisation (issue 2): Current webextension APIs lack a reliable and straightforward way to inject scripts modifying the JavaScript environment before page scripts start running. As JShelter (and Chrome Zero) allows configurable protection that may differ per origin, it needs to access an asynchronous API that stores the configuration. Hence, a naïve implementation adds additional delay that may allow page scripts to access original, unprotected API calls. Note that once page scripts have the opportunity to access the original API implementation, they can store the unpro-

tected version. There is no way (for a webextension) to reverse the leak.

3. Missed context (issue 3): Chrome Zero does not apply protection in iframes and worker threads.

In addition, Firefox suffers from a long-standing unfixed bug [41] that prevents up to 10 % of Firefox webextensions from working correctly on pages whose Content Security Policy (CSP) forbids inline scripts [25] (issue 4).

Preventing issue 1 is simple. The JShelter developer needs to identify the correct method to protect. For example, web developers know that they can get precise timestamps by calling `performance.now()`. But the method `now` is not a method of the `performance` object but it is rather available through prototype chain from `Performance.prototype`. We tackle this issue in two steps. (1) Before we implement a protection, we analyse the prototype chain and pick the correct object implementing the property or method to wrap. Left alone, this approach is brittle: it can be broken by changes in the DOM APIs specification or by browser implementation. Therefore JShelter applies an additional step. (2) The injection code checks at runtime whether the property (or method) is actually implemented as an own property by the object defined in step 1 or if the property is inherited. In the latter case, JShelter traverses the prototype chain until it finds and replaces the correct property, overriding step 1 choice.

A significant effort of the JShelter development went into developing a reliable cross-browser early script injection that tackles issues 2–4. These are problems affecting several privacy and security webextensions. In fact, the techniques adopted and further honed while developing JShelter had been extracted from the NoScript Security Suite[21], refactored into the NSCL and made publicly available for reusing and contributing back. This aims to minimise the maintenance burden and mitigate the danger of introducing insidious bugs and security vulnerabilities due to feature mismatches and multiple code paths by abstracting the common functionality shared among security and privacy webextensions, providing consistent implementations across multiple browser engines and shielding developers from the browser-dependent implementation details.

The NSCL tackles issue 2 in its `DocStartInjection` module[22], by preprocessing URL-dependent configuration inside a `BeforeNavigate` event handler. This event is fired every time the browser starts loading a new page and notifies the webextension of the destination URL, which JShelter uses to build a configuration object in advance and make it available to the content script before it starts its own processing. This technique always succeeds in pre-configuring the webextension on document start, before any page script can run, on Firefox; but on Chromium it might sometimes fail due to race conditions. As a safety net for these edge cases, when no configuration object is found by the content script, the special `SyncMessage` API[23], an ugly (because based on the deprecated synchronous `XMLHttpRequest` API) but effective hack provided by the NSCL, is used to still retrieve the correct settings in a timely manner.

To address issue 3, the configuration of the webextension (`manifest.json`) registers code injection into all the newly created windows, including subframes. Unfortunately, this alone cannot prevent dynamically created windows and frames from being exploited by the originator page to retrieve pristine unwrapped objects and therefore work around the intended protections: `window.open()`, `contentWindow`, and `contentDocument.window` allow access to a new window object immediately after its creation (synchronously); before any initialisation (including the injection of webextension content script) occurs. To fix this problem, NSCL `patchWindow()` API modifies `window.open()`, `contentWindow`, and `contentDocument.window` to recursively wrap the newly created window just before it gets returned.[24]. A further often overlooked possibility to access unwrapped APIs is the historical legacy of subframe windows of all kinds being also immediately available at creation time by just indexing their parent window as an unwrappable pseudo array (e.g. `window[0]` is a synonym of `window.frames[0]`). The NSCL takes care of this problem by automatically patching all not yet patched `window[n]` objects every time the DOM structure is modified, potentially creating new windows. This requires accounting for all methods and accessors by

21 https://noscript.net/

22 https://github.com/hackademix/nscl/20220330/main/service/DocStartInjection.js

23 https://github.com/hackademix/nscl/20220330/main/common/SyncMessage.js

24 https://github.com/hackademix/nscl/blob/20220330/content/patchWindow.js#L247

which the DOM can be changed in JavaScript and wrapping them[25]. Regarding web workers, JShelter either disables or emulates them in the main thread (inheriting the wrapped JavaScript environment), depending on the protection level chosen by the user. The NSCL provides a third option: wrapping workers by injecting the wrappers in their own browser context via its `patchWorkers()` API. The implementation is very complex and still experimental[26] [27]. It needs more testing before it can be confidently deployed to a general audience.

Finally, NSCL works around issue 4 (script injection failing on Firefox for some pages protected by CSP), by leveraging a Firefox-specific privileged API meant to safely share functions and objects between page scripts and WebExtensions[28]. On Chromium, where such API is not available but injected scripts have no special powers and therefore do not need those safety measures, NSCL provides shims to expose a uniform interface for injected code and reduce the burden of cross-browser development.

5 Evaluation

JShelter is available as a webextension in addons.mozilla.org, Chrome Store, and Opera Store from early development stages. We employ the release early, release often strategy, but we do not release early if we are concerned about possible security bugs in the new version.

5.1 Fingerprinting inconsistencies

We are aware that a fingerprinter may observe some inconsistencies. For example, JShelter modifies each read canvas. Should the page scripts probe a single-colour-filled canvas, JShelter would introduce small changes in some pixels. Hence, a page script might learn that protection against canvas fingerprinting is in place.

A naïve implementation available in earlier JShelter versions modified all canvases of the same size in the same way. Hence a fingerprinter could have created two canvases, one for the fingerprinting and the other to learn what pixels are modified and consequently revert the modifications. We removed the vulnerability before anyone outside our team discovered the issue. Nevertheless, the little lies modifications (see §4.2.1) have a performance hit. For all APIs that allow obtaining hardware-rendered data like the Canvas, WebGL, and WebAudio APIs, JShelter needs to access all data in two iterations, first to create a hash that controls the modifications in the second iteration. Hence, the same content is deterministically modified the same way, and different content is modified differently.

Consider `AudioBuffer.prototype.getChannelData` that should allow quick access to pulse-code modulation audio buffer data without data copy. A fingerprinter might be interested in a couple of samples only. However, the spoofing mechanism needs to access all data, so the method is much slower (learning that the time of `getChannelData` takes too long is usable for fingerprinting). During testing, we encountered ebay.com, which is not responsible in current JShelter versions without adblocker due to too much time spent in the spoofing code.

We are not aware of any isolated side-effect that reveals JShelter. For example, some similar webextensions do not modify `toString`. A page script could detect such a webextension as each webextension modifying the call by the same technique will likely use a different code. Nevertheless, we are aware and do not hide that users of JShelter are vulnerable to focused attacks. Our goal is to offer a protection indistinguishable from another privacy-improving tool for each modified API. Nevertheless, a focused observer will very likely be always able to learn that a user is using JShelter if they aggregate the observable inconsistencies of all APIs produced by JShelter.

5.2 Timing events

Biometrics cannot be forgotten or stolen [28]. However, JShelter forges timestamps from all JavaScript timestamp sources consistently. As the biometric feature computation is based on time tracking [28, 47, 48, 70]. Forged timestamps result in fake biometrical data.

JShelter implements rounding and, by default, randomises the timestamps as Chrome Zero does [39]. In comparison, Firefox Fingerprinting Protection and Tor-

25 https://github.com/hackademix/nscl/blob/20220330/content/patchWindow.js#L311
26 https://github.com/hackademix/nscl/blob/20220330/content/patchWorkers.js
27 https://github.com/hackademix/nscl/blob/20220330/service/patchWorkers.js
28 https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Sharing_objects_with_page_scripts

browser implement only rounding, which makes the technique visually easily detectable. Compared with Chrome Zero, JShelter modifies all APIs that produce timestamps, including events (see threat T1), geolocation, gamepads, virtual reality and sensors.

Computer clocks do not measure time accurately, but each has a built-in error. Previous research [30, 51, 56] established that such errors are unique to a device and observable on the network. Jireš [27] studied the influence of timestamp rounding (Torbrowser, Firefox Fingerprinting Protection) and rounding and randomisation (JShelter, Chrome Zero). He computed clock skew from rounded timestamps, but he could not remove the noise from rounded and randomised timestamps. However, this result should be validated; long-lasting (at least tens of minutes) measurements might remove the randomisation noise and reveal the clock skew. Nevertheless, Polčák and Franková [51] observed that timestamps provided by JavaScript are affected by time synchronisations (such as NTP). Hence, we advise to combining JShelter round and randomisation with continuous-time synchronisation to hide built-in clock skew.

5.3 Sensors

We discovered a loophole in the `Sensor.timestamp` attribute[29]. The value describes when the last `Sensor.onreading` event occurred in millisecond precision. We observed that the time origin is not the time of browsing context creation but the last boot time of the device. Exposing such information is dangerous as it allows to fingerprint the user easily. It is unlikely that two different devices will boot at the same time.

JShelter protects the device by provisioning the time since the browser created the page context (the same value as `performance.now()`. Such timestamps uniquely identify the reading without leaking anything about a device. Future work can determine if such behaviour appears in the wild. If all devices and browsers incorporate the loophole, we should provide a random boot time.

Figure 5 shows readings from a real and fake magnetometer. The left part (a) shows a stationary device. The magnetic field is not stable due to small changes in Earth's magnetic field and other noise. The middle part of the figure (b) shows a device that changed its position several times during the measurement. We analysed traces of sensors readings collected in various locations and environments. Fig. 5 (c) shows readings generated by JShelter fake magnetometer. The values look like actual sensor readings. Nevertheless, the generator uses a series of constants whose optimal values should be a subject of future research and improvements.

5.4 Fingerprint Detector effectivity

The FPD heuristics were designed to keep the number of false positives as low as possible. As FPD can optionally block all subsequent requests by a fingerprinting page and JShelter provides complementary protections, FPD blocks only indisputable fingerprinting attempts. We conducted real-world testing of FPD and refined its detection heuristics accordingly.

In terms of testing methodology, we manually visited homepages and login pages of the top 100 websites from the Tranco list[30]. Inaccessible websites were randomly replaced by websites from the top 200 list. Before visiting a website, we wiped browser caches and storage to remove previously-stored identifiers. Hence, the visited pages may have deployed fingerprinting scripts more aggressively to identify the user and reinstall the identifier.

To boost the probability of fingerprinting even more, we switched off all protection mechanisms offered by the browser. However, we blocked third-party cookies because our previous experience suggests that the missing possibility to store a permanent identifier tempts trackers to start fingerprinting. We repeated the visits with both Google Chrome and Mozilla Firefox.

We used FPMON and DFPM webextensions to create the ground truth. For each visited page, we computed its fingerprinting score. FPMON reports fingerprinting pages with colour. We assigned yellow colour 1 point and red colour 3 points. DFPM reports danger warnings. If DFPM reports one danger warning, we assign 1 point to the page. For a higher number of danger warnings, we assign 3 points to the page. Therefore, each page gets a fingerprinting score from 0 to 6.

29 Tested with Samsung Galaxy S21 Ultra; Android 11, kernel 5.4.6-215566388-abG99BXXU3AUE1, Build/RP1A.200720.012.G998BXXU3AUE1, Chrome 94.0.4606.71 and Kiwi (Chromium) 94.0.4606.56 and Xiaomi Redmi Note 5; Android 9, kernel 4.4.156-perf+, Build/9 PKQ1.180901.001, Chrome 94.0.4606.71

30 https://tranco-list.eu/list/23W9/1000000

(a) Stationary device (b) Moving device (c) Fake readings

Fig. 5. Magnetometer readings.

	Ground truth (FPMON + DFPM)	JShelter (FPD)	False positives (FPD)	False negatives (FPD)
Homepages	**20**	**20**	**1**	**0**
Login pages	**34**	**30**	**1**	**7**

Table 1. Results of FPD study from the manual crawl of the top 100 web pages according to the Tranco list.

Table 1 shows the results of the comparison. We classify a page as fingerprinting when its score is 4 or more. We did not count pages with a score of 3 or 2 as fingerprinting because their status is unclear. We consider scores 0 and 1 as an indication of no fingerprinting. However, as reported above, the ground truth is far from flawless. We encountered many exceptions during testing and examined them in detail. In many cases, FPD detects fingerprinting, but the reference webextensions do not.

(1) *The score of 6.* FPD successfully detects pages with a score of 6 except the *Google login* page (which occurred six times in the testing set). We studied the accessed APIs and decided that accessed APIs do not provide enough entropy for most users. Nevertheless, we count *Google login* pages as false negatives.

(2) *The score of 4.* FPD classifies these web pages as fingerprinting with two exceptions, *Facebook login* page and `yandex.ru`. Both are borderline cases that do not obtain enough entropy.

(3) *The score of 3.* These pages are borderline cases. FPD detected fingerprinting only on one of these pages, the *Paypal login* page. The detection was correct as we manually found clear tracks of canvas fingerprinting.

(4) *The score of 2.* We assume that web pages with this score learn a fingerprint that is likely short on entropy. FPD detected two web pages with this score as fingerprinters, namely *Cloudflare login* page and *Washington Post login* page. A closer analysis revealed that both pages use canvas fingerprinting in and other fingerprinting methods.

(5) *The score of 1 or 0.* FPD detected fingerprinting on `ebay.com`. Manual inspection showed that `ebay.com` did indeed fingerprint using canvas fingerprinting, audio fingerprinting and other techniques.

The asymmetry between detection on different browsers was minor and had minimal impact on detection. Moreover, FPD automatically recalculates heuristics if a browser does not support a monitored API.

5.5 Network Boundary Shield

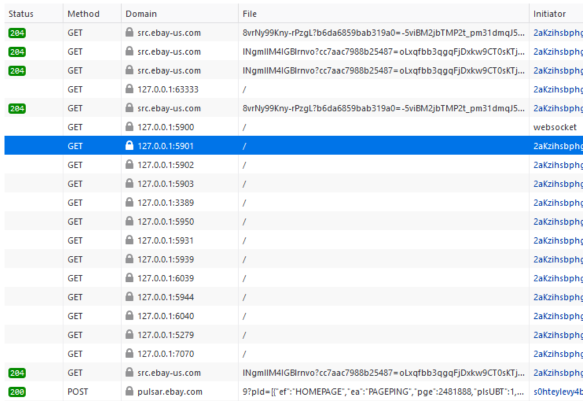
5.5.1 Localhost scanning

Some web pages, like ebay.com, scan (some users) for open local TCP ports to detect bots with open remote desktop access or possibly to create a fingerprint. The web page instructs the browser to connect to the *localhost* (`127.0.0.1`) and monitors the errors to detect if the port is opened or closed. See Fig. 6 for an example.

When we developed NBS we did not anticipate localhost port scanning. When we first encountered the eBay port scanning case, we knew that this behaviour should trigger NBS as the requests cross network boundaries. We accessed ebay.com, detected the scanning by Web Developer Tools (Fig. 6) and checked that NBS is indeed triggered and works as expected.

As Article 29 Working Party clarified [69, use case 7.5], user-centric security can be viewed as strictly necessary to provide the service. So it seems likely that port scanning for security reasons would trigger the ePrivacy exception and user consent is not necessary.

As port scanning is a part of the login mechanism, open ports are personal data without a doubt. So GDPR also applies. GDPR also list security as a possible legitimate interest of a data controller (e.g. eBay), see



Status	Method	Domain	File	Initiator
204	GET	src.ebay-us.com	8vrNy99Kny-rPzgL?b6da6859bab319a0=-5viBM2jbTMP2t_pm31dmqJ5...	2aKzihsbph
204	GET	src.ebay-us.com	INgmlIM4IGBIrnvo?cc7aac7988b25487=oLxqfbb3qgqFjDxkw9CT0sKTj...	2aKzihsbph
204	GET	src.ebay-us.com	INgmlIM4IGBIrnvo?cc7aac7988b25487=oLxqfbb3qgqFjDxkw9CT0sKTj...	2aKzihsbph
	GET	127.0.0.1:63333	/	2aKzihsbph
204	GET	src.ebay-us.com	8vrNy99Kny-rPzgL?b6da6859bab319a0=-5viBM2jbTMP2t_pm31dmqJ5...	2aKzihsbph
	GET	127.0.0.1:5900	/	websocket
	GET	127.0.0.1:5901	/	2aKzihsbph
	GET	127.0.0.1:5902	/	2aKzihsbph
	GET	127.0.0.1:5903	/	2aKzihsbph
	GET	127.0.0.1:3389	/	2aKzihsbph
	GET	127.0.0.1:5950	/	2aKzihsbph
	GET	127.0.0.1:5931	/	2aKzihsbph
	GET	127.0.0.1:5939	/	2aKzihsbph
	GET	127.0.0.1:6039	/	2aKzihsbph
	GET	127.0.0.1:5944	/	2aKzihsbph
	GET	127.0.0.1:6040	/	2aKzihsbph
	GET	127.0.0.1:5279	/	2aKzihsbph
	GET	127.0.0.1:7070	/	2aKzihsbph
204	GET	src.ebay-us.com	INgmlIM4IGBIrnvo?cc7aac7988b25487=oLxqfbb3qgqFjDxkw9CT0sKTj...	2aKzihsbph
200	POST	pulsar.ebay.com	9?pld=[{"ef":"HOMEPAGE","ea":"PAGEPING","pge":2481888,"plsUBT":1,...	s0hteylevy4

Fig. 6. eBay webpage scanning the local computer for open ports.

recital 49. Nevertheless, if such a scan is proportionate is an open question; the legitimate interests of data controllers (such as eBay) may be overridden by the interests or fundamental rights and freedoms of the data subject (JShelter users), see GDPR, Article(6)(1)(f).

Nevertheless, Article 12-14 of GDPR lists requirements on the information that a data controller should reveal to each data subject before the data processing starts or in a reasonable time afterwards. Hence, each controller employing such port scanning should reveal, for example, in the privacy policy, what categories of data it is using and for which purposes. Several web articles covering the eBay case[31] suggests that eBay and its processor ThreatMetrix are secretive about data being collected.

Another GDPR issue might be data transfers to third countries. If the information leaves EEA, data transfers of open ports may not be compatible with GDPR in light of the CJEU C-311/18 decision.

Hence, JShelter can be used by data protection authorities to detect local port scanning and check if the data controller informs data subjects in line with GDPR requirements.

5.5.2 Comparison with Private Network Access

Recently, Google announced Private Network Access (PNA)[32] that should become W3C standard[33]. PNA solves the same problem as NBS, but the solution is different. PNA-compatible browsers send HTTP Requests to the local networks with the additional header: `Access-Control-Request-Private-Network: true`. The local resource can allow such access with HTTP reply header `Access-Control-Allow-Private-Network: true`. If it does not, the browser blocks the access.

NBS works differently. Firefox version leverages DNS API to learn that a public web page tries to access the local network and blocks the request before the browser sends any data. Chromium-based browsers do not support DNS API, so the first request goes through. NBS learns the IP address during the reply processing. NBS blocks any future request before it is made once it learns the IP address during the reply processing. Hence, NBS limits the network bandwidth and prevents any state modification on a local node that may be caused by request going through, except for the learning phase in Chromium-based browsers. We consider both approaches to solving threat T5; it is up to the user what solution they prefer.

Note that Google postponed the Chrome PNA deployment during March 2022, so Chrome users without JShelter or another webextension with similar capabilities are not protected.

5.6 Feedback from users

Some users found JShelter immediately after initial upload to webextension sites. Nevertheless, the number of users increased massively only after an announcement by Free Software Foundation. Figure 7, shows JShelter users in time in Firefox and Chrome. The graph shows that JShelter has an audience and users want to control their browsers.

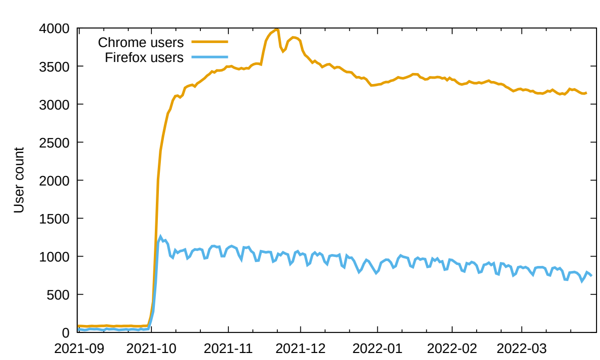


Fig. 7. Number of JShelter users.

31 https://blog.avast.com/why-is-ebay-port-scanning-my-computer-avast, https://www.theregister.com/2020/05/26/ebay_port_scans_your_pc/

32 https://developer.chrome.com/blog/private-network-access-prefilght/

33 https://wicg.github.io/private-network-access/

Based on the feedback from users, one of the reasons for the decline is that they encounter too many broken or slow pages. We do not think that JShelter is finished and fixing broken pages was not a priority yet.

Another reason is that users do not understand the little lies fingerprint prevention. They want to hide in the crowd (see §3.3). The most controversial of which is WebGL vendor, unmasked vendor, renderer, and unmasked renderer spoofing. We do not know any list of real-world strings, and even if we knew, we are not sure if we could avoid inconsistencies. Hence we decided that the threat model defending from a fingerprinter not focused on revealing JShelter users allows for the generation of random strings per origin and session for the little lies JSS profile (see §4.2). Some users do not understand the explanation even though we highlight that similar randomly generated strings are already available through `MediaDevices.prototype.enumerateDevices`, the created profile is unique by design.

A common problem is that users do not understand what JShelter is doing and that several modules work in parallel and can be enabled and configured separately. We tweaked the UI several times to make the UI as straightforward as possible and we added explanations and want to add even more explanations (for example, to the popup window).

JShelter users also reported false-positive NBS detections when using DNS-based filtering programs. Some DNS-based filters return the localhost IP address for any blocked domain. In that case, NBS correctly detects that a public page tries to access a local resource, blocks the request, and notifies the user. Users often do not understand that their DNS configuration is broken, and the DNS blocker should be reconfigured to return 0.0.0.0 (invalid address). This behaviour triggered another observation that NBS generated too many notifications. Some users do not want to be notified at all. We added an option to turn off notifications. We limited the number of notifications if they are enabled and the web page accesses local nodes during a short time frame. Additionally, we added explanation texts. It seems that users do not report issues with NBS notifications anymore.

Many privacy-related webextensions report the number of blocked elements in the badge icon. Previous research projects like Chrome Zero depicts currently applied protections. Early JShelter versions reported applied level as well, but the feedback preferred showing the number of blocked elements and using colours. We decided to (1) report the number of accessed API groups and (2) report the likelihood of fingerprinting as a colour starting from shades of green through yellow to shades of red. Figure 8 shows examples of badged icons that received positive reception.

Fig. 8. Interactive badge icons.

Early versions spoofed information sent in HTTP headers. We removed this option as it broke pages, created inconsistencies [32], and needed too much maintenance to update browser versions. Nevertheless, passive fingerprinting is common, see Fig. 1 and users are asking about such a feature. We will likely revisit the early development decision and consider adding a passive fingerprint shield.

6 Conclusion

Previous research established that browser security, privacy, and customizability are important topics [6, 13, 20, 32, 38, 39, 51, 66]. The imminent danger of third-party cookies' removal forces trackers to employ even more privacy-invading techniques. Real-time bidding leaves users as easy targets for various attacks, including gaining information about other applications running on the local computer [46]. Moreover, continuous additions of new JavaScript APIs open new ways for fingerprinting the browsers and gaining additional knowledge about the browser or user preferences and physical environment. One of the major concerns is a lack of effective tools that everyday user wants to use. Current methods to tackle web threats are list-based blockers that might be evaded with a change of URL, specialised browsers, or research-only projects that are quickly abandoned.

In contrast, JShelter is a webextension that can be installed on major browsers and consequently does not require the user to change the browser and routines. We integrate several previous research projects like Chrome Zero [39], little-lies-based fingerprinting prevention [44, 50], and ideas of limiting APIs brought by Web API Manager [61]. JShelter comes with a heuristic-based fingerprint detector and prevents webpages from misusing the browser as a proxy to access the local network and computer. We needed to solve issues with reliable environment modifications that stem from webextension API that opens many loopholes that previous research exploited [57]. In addition to JShelter, we introduced NSCL. Both NoScript Security Suite and JShelter include NSCL. Moreover, NSCL is available for other privacy- and security-related webextensions.

In cooperation with the Free Software Foundation, we aim for long-term JShelter development; thus, users' privacy and security should be improved in the future. We explain fingerprinting vectors introduced by Sensor API in mobile browsers. Data protection specialists should detect browser fingerprinting and other information leaks with JShelter. We integrated fingerprint report and notifications to facilitate the task. We discussed considerations and issues connected with deployment. The webextension is under development. Future work will include fixing problems breaking pages, improved heuristics of FPD, and research fingerprinting on login pages. We want to revisit and evaluate the little-lies-based anti-fingerprinting technique; are the little changes enough to stop a determined fingerprinter that can, for example, approximate colour values of several pixels or repeat an effect multiple times? JShelter should not be considered a single bullet-proof solution. We anticipate that everyday users will install JShelter together with other webextensions like list-based blockers or JavaScript blockers.

7 Acknowledgement

This project was funded through the NGI0 PET Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 825310 as JavaScript Restrictor and JShelter projects. This work was supported in part by the Brno University of Technology grant FIT-S-20-6293. We thank Martin Bednář for capturing Figure 6.

References

[1] Administrative fine - Grindr LLC, 2021. URL https://www.datatilsynet.no/contentassets/8ad827efefcb489ab1c7ba129609edb5/administrative-fine---grindr-llc.pdf.

[2] Gunes Acar, Steven Englehardt, and Arvind Narayanan. No boundaries for user identities: Web trackers exploit browser login manager, 2017. Available online at https://freedom-to-tinker.com/2017/12/27/no-boundaries-for-user-identities-web-trackers-exploit-browser-login-managers/.

[3] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 674–689. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2957-6.

[4] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: Dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1129–1140. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-2477-9.

[5] Akram Bayat, Amirhossein Bayat, and Sina Amir. Classifying human walking patterns using accelerometer data from smartphone. 12 2017.

[6] John Bergbom. Attacking the internal network from the public internet using a browser as a proxy, 2019. Forcepoint research report available at https://www.forcepoint.com/sites/default/files/resources/files/report-attacking-internal-network-en_0.pdf.

[7] Andrew Bortz and Dan Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 621–628. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-654-7. URL http://doi.acm.org/10.1145/1242572.1242656.

[8] Brave. Updates & timeline for brave's work to fix "rtb" adtech, 2019. URL https://brave.com/rtb-updates/. Visited 2019-12-16.

[9] Yinzhi Cao, Song Li, and Erik Wijmans. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *Proceedings of Network & Distributed System Security Symposium (NDSS)*, 2017.

[10] Autorité de protection des données Gegevensbeschermingsautoriteit. Decision on the merits 21/2022 of 2 february 2022, 2022. URL https://www.autoriteprotectiondonnees.be/publications/decision-quant-au-fond-n-21-2022-english.pdf. Unofficial translation from Dutch.

[11] A. Decan, T. Mens, and E. Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 181–191, 2018. ISSN 2574-3864.

[12] Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, volume 6205 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, DE, 2010. ISBN 978-3-642-14526-1.

[13] Steven Englehardt, Gunes Acar, and Arvind Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. *Proceedings on Privacy Enhancing Technologies*, 2020:220–238, 2020.

[14] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1388–1401. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4139-4.

[15] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*,

CCS '00, pages 25–32. ACM, New York, NY, USA, 2000. ISBN 1-58113-203-4. URL http://doi.acm.org/10.1145/352600.352606.

[16] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security*, pages 107–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-47854-7.

[17] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: An analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 309–318. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 2018. ISBN 978-1-4503-5639-8. URL https://doi.org/10.1145/3178876.3186097.

[18] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 245–261, 2018. ISSN 2375-1207.

[19] Daniel Gruss, David Bidner, and Stefan Mangard. Practical memory deduplication attacks in sandboxed javascript. In *Computer Security – ESORICS 2015*, pages 108–122. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24174-6.

[20] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 300–321. Springer International Publishing, Cham, 2016. ISBN 978-3-319-40667-1.

[21] Gabor Gyorgy Gulyas, Doliere Francis Some, Nataliia Bielova, and Claude Castelluccia. To extend or not to extend: On the uniqueness of browser extensions and web logins. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, WPES'18, pages 14–27. ACM, New York, NY, USA, 2018. ISBN 978-1-4503-5989-4.

[22] Jun Han, Emmanuel Owusu, Le T. Nguyen, Adrian Perrig, and Joy Zhang. Accomplice: Location inference using accelerometers on smartphones. pages 1–9, 01 2012.

[23] Kashmir Hill. Be warned: Customer service agents can see what you're typing in real time, 2018. Available online at https://gizmodo.com/be-warned-customer-service-agents-can-see-what-youre-t-1830688119.

[24] ICO — Information Commissioner's Office. Update report into adtech and real time bidding, 2019. URL https://ico.org.uk/media/about-the-ico/documents/2615156/adtech-real-time-bidding-report-201906.pdf. visited 2019-12-16.

[25] Bohdan Inhliziian. Impact of the application of the content-security-policy header on firefox webextensions, 2020. URL https://www.fit.vut.cz/study/thesis/22483/. Bachelor's thesis, Brno University of Technology, Faculty of Information Technology.

[26] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *IEEE Symposium on Security & Privacy*, 2021.

[27] Michal Jireš. Computer identification using javascript timestamps, 2020. URL https://www.fit.vut.cz/study/thesis/22308/. Bachelor's thesis.

[28] Zach Jorgensen and Ting Yu. On mouse dynamics as a behavioral biometric for authentication. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, page 476–482. Association for Computing Machinery, New York, NY, USA, 2011. ISBN 9781450305648.

[29] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, abs/1801.01203, 2018.

[30] Tadayoshi Kohno, Andre Broido, and Kimberly C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005. ISSN 1545-5971.

[31] Georgios Kontaxis and Monica Chew. Tracking protection in firefox for privacy and performance. In *Web 2.0 Security & Privacy Workshop*, 2015.

[32] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. volume 14. Association for Computing Machinery, New York, NY, USA, apr 2020. ISSN 1559-1131. URL https://doi.org/10.1145/3386040.

[33] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 878–894, 2016.

[34] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William K. Robertson, Christo Wilson, and Engin Kirda. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. *CoRR*, 2018.

[35] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 2016.

[36] Célestin Matte, Nataliia Bielova, and Cristiana Santos. Do cookie banners respect my choice? measuring legal compliance of banners from iab europe's transparency and consent framework, 2019. ArXiv eprint 1911.09964, available at https://arxiv.org/abs/1911.09964, last access 2019-12-13.

[37] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *2012 IEEE Symposium on Security and Privacy*, pages 413–427, 2012. ISSN 1081-6011.

[38] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 319–333, 2017.

[39] Moritz Lipp Michael Schwarz and Daniel Gruss. Javascript zero: Real javascript and zero side-channel attacks. In *Network and Distributed Systems Security Symposium 2018*, 2018. ISBN 1-1891562-49-5.

[40] Keaton Mowery and Hovav Shacham. Pixel Perfect: Fingerprinting Canvas in HTML5. In *Proceedings of W2SP*, 2012.

[41] Mozilla Bugzilla. [meta] page csp should not apply to content inserted by content scripts (v2 issue). URL https:

//bugzilla.mozilla.org/show_bug.cgi?id=1267027. last visit 2022-03-30.

[42] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. ScriptProtect: Mitigating unsafe third-party javascript practices. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, page 391–402. Association for Computing Machinery, New York, NY, USA, 2019. ISBN 9781450367523.

[43] Erik Trickela nad Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupé. Everyone is different: Client-side diversification for defending against extension fingerprinting.

[44] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. PriVaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 820—-830. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2015. ISBN 9781450334693.

[45] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*, pages 541–555, 2013. ISSN 1081-6011.

[46] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1406–1418. Association for Computing Machinery, New York, NY, USA, 2015. ISBN 9781450338325.

[47] Avar Pentel. High precision handedness detection based on short input keystroke dynamics. In *2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*, pages 1–5, 2017.

[48] Avar Pentel. Predicting age and gender by keystroke dynamics and mouse patterns. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, UMAP '17, pages 381—-385. Association for Computing Machinery, New York, NY, USA, 2017. ISBN 9781450350679.

[49] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. Technical report, 2010. Version 0.34, Available online at https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf.

[50] Vikas Mishra Pierre Laperdrix, Benoit Baudry. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *9th International Symposium on Engineering Secure Software and Systems*, page 17, 2017.

[51] Libor Polčák and Barbora Franková. Clock-skew-based computer identification: Traps and pitfalls. *Journal of Universal Computer Science*, 21(9):1210–1233, 2015. ISSN 0948-6968.

[52] Johnny Ryan. Report from Dr Johnny Ryan – behavioural advertising and personal data, 2018. URL https://brave.com/wp-content/uploads/2018/09/Behavioural-advertising-and-personal-data.pdf.

[53] Zach Schneider. event-stream vulnerability explained, 2018. Available online at https://schneider.dev/blog/event-stream-vulnerability-explained/.

[54] Michael Schwarz, Florian Lackner, and Daniel Gruss. Javascript template attacks: Automatically inferring host information for targeted exploits. In *Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[55] Michael Schwarz, Florian Lackner, and Daniel Gruss. Latex gloves: Protecting browser extensions from probing and revelation attacks. In *Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[56] Swati Sharma, Alefiya Hussain, and Huzur Saran. Experience with heterogenous clock-skew based device fingerprinting. In *Workshop on Learning from Authoritative Security Experiment Results*, pages 9–18. ACM, 2012. ISBN 978-1-4503-1195-3. Arlington, Virginia.

[57] Anatoly Shusterman, Ayush Agarwal, Sioli O'Connell, Daniel Genkin, Yossi Oren, and Yuval Yarom. Prime+Probe 1, JavaScript 0: Overcoming browser-based Side-Channel defenses. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2863–2880. USENIX Association, August 2021. ISBN 978-1-939133-24-3.

[58] David Silver, Suman Jana, Dan Boneh, Eric Chen, and Collin Jackson. Password managers: Attacks and defenses. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 449–464. USENIX Association, San Diego, CA, 2014. ISBN 978-1-931971-15-7.

[59] Michael Smith, Craig Disselkoen, Shravan Narayan, Fraser Brown, and Deian Stefan. Browser history re:visited. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. USENIX Association, Baltimore, MD, 2018. URL https://www.usenix.org/conference/woot18/presentation/smith.

[60] Peter Snyder, Lara Ansari, Cynthia Taylor, and Chris Kanich. Browser feature usage on the modern web. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 97–110. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4526-2. URL http://doi.acm.org/10.1145/2987443.2987466.

[61] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 179–194. ACM, New York, NY, USA, 2017. ISBN 978-1-4503-4946-8. URL http://doi.acm.org/10.1145/3133956.3133966.

[62] O. Starov and N. Nikiforakis. Xhound: Quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 941–956, 2017. ISSN 2375-1207.

[63] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. Are you sure you want to contact us? quantifying the leakage of pii via website contact forms. volume 2016, pages 20–33, 2016.

[64] Ben Stock and Martin Johns. Protecting users against xss-based password manager abuse. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 183–194. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2800-5. URL http://doi.acm.org/10.1145/2590296.2590336.

[65] Paul Stone. Pixel perfect timing attacks with HTML5, 2013. Black Hat 2013. Context Information Security whitepaper available online at https://www.contextis.com/en/resources/white-papers/pixel-perfect-timing-attacks-with-

html5.

[66] Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1382–1393. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3832-5.

[67] Tom Van Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In Juan Caballero, Eric Bodden, and Elias Athanasopoulos, editors, *Engineering Secure Software and Systems*, pages 106–121. Springer International Publishing, Cham, 2016. ISBN 978-3-319-30806-7.

[68] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In *Proceedings of the 27th USENIX Security Symposium*. Baltimore, United States, August 2018. URL https://hal.inria.fr/hal-01820197.

[69] WP29 — Article 29 Data Protection Working Party. Opinion 9/2014 on the application of Directive 2002/58/EC to device fingerprinting, 2014. WP224, https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp224_en.pdf.

[70] Nan Zheng, Aaron Paloski, and Haining Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 139–150. Association for Computing Machinery, New York, NY, USA, 2011. ISBN 9781450309486.