Libor Polčák*, Martin Bednář, Marek Saloň, Giorgio Maone, and Radek Hranický

JShelter: Give Me My Browser Back

Abstract: Please put abstract here.

Keywords: keywords, keywords

1 Introduction

Most of the people interact with web pages on daily basis. Nowadays, many activities are often carried via a web browser including shopping, searching for travel information, doing business and office work, performing leisure activities such as gaming. For several years, browser vendors keep adding new JavaScript APIs to enable development of richer web apllications [**?**].

Some of the recently added APIs influence the privacy of the users. For example, the geolocation API[1] is useful when a user searches maps or navigates in the real world. In this cases, the user is willing to share the location. However, the users might not be willing to share the location with all sites they are visiting. In the case of geolocation APIs, browsers ask users for permission but not all APIs asks users for permissions. Moreover, even in the geolocation API case, the users can grant permanent access to a preferred map service but sometimes they want to share more precise location (e.g. during navigation) and other times they want to share the location with a limited precision (e.g. they are searching for a location not related to their current position).

This paper presents a web browser extension called JShelter that provides the opportunity to tweak the APIs that the browser provides to the web pages. To do so, we study the JavaScipt code constructs that can be leveraged to tweak the APIs according to the user current needs. We implemented JShelter for Firefox, and Chromium-based browsers like Chrome, Opera, and Edge.

Evaluation ...

This paper is organized as follows. Section 2 presents the threats that the users face during web browsing. Section 3 compares the extension described in this paper to other security and privacy related extensions. Section 4 provides the design decisions that we faced during the development of JShelter. Section 5 reports on the success of the extension. Section 6 discusses the impact of this work and Section 7 concludes this paper.

2 Threats

This section presents threats that every web user faces everytime they load and execute unknown JavaScript code. Although modern browsers JavaScript engines employ security measures, such as same-origin policy[2], there are still threats that are not mitigated.

2.1 T1: Detail user behaviour monitoring

In theory, law like GDPR and ePrivacy Regulation give each person control over their personal data and devices. In practise, there is a significant lack of control of personal data on web [7, 22, 32, 33, 42]. The advertisement technologies are under a big scrutiny in Europe [**? ?**] but tracking scripts are omnipresent on the web. Users risk complete revelation of their browsing history.

Web content providers want detail information on user interaction with the web pages. JavaScript event listeners and handlers can track user activities such as mouse movement, typing, clicking etc. [11]. The web site operator can replay the user session in real time or later. Customer services provide chat windows providing information in real time. However, some libraries

***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
Martin Bednář: Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic, E-mail: ibednar@fit.vut.cz
Marek Saloň: Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno
Giorgio Maone: InformAction, 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

1 https://developer.mozilla.org/en-US/docs/Web/API/Geolocation

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

for the chat interaction transfers the question as it is typed instead of waiting for the user to press the send button [21].

2.2 T2: Browser and computer fingerprinting

Historically, T1 tracking was performed using third party cookies. 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 a set of features that make (almost) every browser uniquely identifiable. [8, 10, 28, 29]. For example, the content of HTTP headers including user agent string, screen size, language, time zone, and system fonts together with hardware-dependant characteristics such as canvas image rendering [8, 36], audio processing [12], installed fonts [14], installed browser extensions [19, 44, 45, 50], the sites that the user is currently logged in [19], clock skew [26, 41] and other techniques [28]. The goal of the fingerprinter is to provide 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 shown that user tracking is becoming more prevelant and complex [31]. 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 extension is to prevent active fingerprinting.

Several studies monitored the deployed fingerprinting techniques on the Internet [2, 3, 12, 15, 39]. Results indicate that evercookies, shared cookies, font enumeration, and canvas are commonly used to identify browsers and consequently their users.

Fortunately, Mozilla is working on integrating some fingerprinting resisting techniques from Tor browser[3], however, the counter-meassures were found to be insufficient [44, 57].

Current research distinguish targeted and not targeted fingerprinting [28]. 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 [44].

Browser fingerprinting can be also 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 your system properties, like the version of the browser. Some websites collect browser fingerprints to distinguish human users and bots to prevent fraud.

2.3 T3: Sensors

Modern (especially portable) devices contain various sensors[4] for reading information about the device position, state, and environment. While the benefits of having sensors are undisputed, allowing websites to access their readings represents a considerable danger.

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 do only work when browser flags like `#enable-experimental-web-platform-features` or `#enable-generic-sensor-extra-classes` are enabled.

Both Generic Sensor W3C Candidate Recommendation Draft[5] and literature mentions several risks like location tracking [20], eavesdropping, keystroke monitoring, device fingerprinting [56], and user identification [4].

2.4 T4: Hostile third party scripts

Whenever a user visit a web page, it can include external scripts using the script element (for example the

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

4 https://w3c.github.io/sensors/

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

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 tracking (T1) is typically performed by third party scripts.

DOM dynamically reflects changes of the page, including password and credit card strings. Several researchers [1, 47, 53] warn that the autofill functionality of password managers can be tricked by hostile scripts to leak user credentials without their awareness. Other research focused on contact forms that leak personal data to unintended recipients [51]. Some packages aim at money stealing [43].

Modern web development includes many libraries. Decan et al. [9] 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 very common that websites use libraries with known security vulnerabilities [30]. Lauinger et al. [30] observed that web sites use unpatched libraries for years. Additionally, they observed that libraries included transitively or via ad tracking code are more likely to be vulnerable as the ecosystem is complex, unorganized and it is often hard to identify the vulnerable package versions. Sometimes, multiple versions of the same library are included simultaneously. Mush et al. [37] 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 1 shows the trend in the number of pages with detected vulnerabilities created by HTTP Archive[6]. It seems that in the last years the pages are becoming less vulnerable but more 58.9 % of pages are vulnerable to at least one known and detected vulnerability.

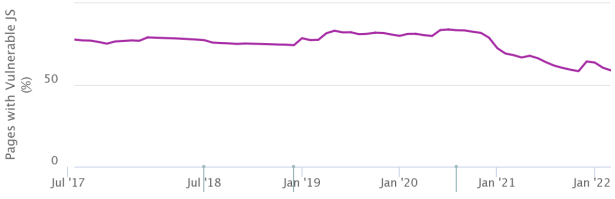

Fig. 1. Web pages with detected vulnerabilities by HTTP archive.

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

2.5 T5: Local network scanning

Devices browsing web are typically connected behind NAT that does not allow external hosts to open connections to devices in local network (e.g. printers). Although the same-origin policy does not allow a web page to access arbitrary resource on any site, there are side-channels that might provide enough information about an existence of a resource, including resources in the local network [5]. The web browser is used as a proxy between the remote web site and resources in local network. Bergbom [5] demonstrated that it possible to execute arbitrary command 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 channels that can reveal what is the user doing with the computer at the moment or was the computer doing recently. For example, content-based page deduplication performed by operating system or a virtual machine hypervisor can reveal if specific images or web sites are currently opened [17] on the same computer (HW) possibly on another virtual machine. The reply time for a specific request depends on the cache content, so the reply time reveals if the content was recently visited [13]. Moreover, even uncached content leaks information on the server state [6]. Bortz and Boneh [6] 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 [54].

Operating systems isolate processes from each other and the kernel. However, deficiencies in hardware can provide possibilities to circumvent the isolation. Gruss et al [18] 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 optimizations and high-precision timings [18]. Later, Gruss et al. [16] showed that industry countermeasures against Rowhammer attacks are ineffective. Recently, Spectre attacks was proven to be executed from JavaScript code [25] and for example leak data in the memory of other processes running on the same system.

Van Goethem et al. [55] employed timing attacks to reveal data about users, for example by measuring the size of the reply for resources with different contents based on users age, gender and location.

Smith et al. [48] determined browser history by the visited link pseudoclass and timining redrawing of the links based on the target URL.

3 Countermeasures

Many popular security and privacy enhancing browser extensions already exists. Let us focus on existing tools addressing the threats raised in Sect. 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 there are many tools focusing on blocking (for example uBlock Origin, EFF Privacy Badger, Ghostery) and also blocklists that are usually compatible with several blockers. Browser like Firefox [27] and Brave include tracking prevention by default. The downside is that it is easy to evade blockers [34]. The malicious web server needs to change the name of the script. For example, one of the Czech bank is currently being investigated for including tracker scripts in their internet banking, the scripts were not matched by the default uBlock Origin block lists. Hence, block lists are very useful as a first line defence and to improve web performance [27]. But blockers are not enough as the niche cases evade the blockers [34].

Extensions like NoScript 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 Sect. 2. But the user needs to evaluate what scripts to allow. HTTP Archive reports[7] 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 content resources from many external sources so such user would need to have a very good knowledge. Moreover, a malicious code may be only a part of a resources, the rest of the resource can be necessary for correct page functionality. So we believe that extensions like NoScript Suite and uMatrix Origin are good but do not protect the user from accidentally allowing a malicious code.

JavaScript Zero [35] (also known as Chrome Zero[8]) expects that a user lets the browser run the vulnerable code and focuses on mitigating T6. Even most skillful 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. [46] have shown that the webpage can obtain access to the original API calls.

Web API Manager [49] classifies JavaScript APIs into 81 standards[9]. The user of the Web API Manager extension can disable all functionality defined by any of the standards. Authors prepared three configurations with standards blocked depending of their benefits and costs [49]. Web API Manager is most effective against T3 and not targeted T2 but it can be useful in mitigating 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 [36]. Additionally, the extension is no longer maintained[10], it is not compatible with Firefox Multi-Account Containers[11], and it suffers from the Firefox bug related to CSP [**?**]. A Web API Manager user with a tailored configuration can potentially be uniquely identified with the JavaScript enumerating code developed by Schwarz et al. [44].

Other vital extensions are cookie managers (threat T1) or local CDN cachers (threats T1 and T4).

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 clients but an IP address of a Tor exit node. Torbrowser is a Firefox fork that tries to make ev-

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

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

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

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

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

ery instance as uniform as possible. For example, every user should browse with the same window size. But a fingerprinter can still learn some information like the underlying operating system [28]. Torbrowser also disables several APIs like WebGL. Consequently, Torbrowser is a very good solution to tackle threats T1, T2, T3, T5, and T6. Nevertheless, Torbrowser user should not install additional extensions to prevent fingerprinting, resize window. As the communication is relayed multiple times by relays spread around the world, both latency and throughput is limited. The list of Tor exit node IP addresses is public. Moreover, Tor is often misused by malicious actors. 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 Linux distribution repositories.

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 [28]. Currently, three sensible approaches to modify fingerprintable values 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 counter-measures. All browsers with the same fingerprint form an anonymity set [40]. 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 changing 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, its cannot be used for cross-domain linking of the same browser.

(3) *Detect and block fingerprint attemps.* As previous approaches targeted mainly fingerprinting mitigation, other countermeasures may take radical steps to provide more definitive protection. An likelihood of ongoing fingerprinting can be detected based on various factors. One of which can be access to APIs that provide sensitive information about browser environment. In case of effective real-time fingerprinting detection, there is an opportunity to deploy anti-fingerprinting measures in time. It is essential to successfully defy any misuse of the gathered fingerprint. First of all, tracking server should not have access to the fingerprint by any means necessary. To reliably prevent sharing the fingerprints with trackers, any network traffic to the tracking server should be blocked. Secondly, it is wise to remove any trace of the fingerprint from a browser to prevent later use. Mechanisms like browser cache, cookies or local storage can be misused to store the fingerprint and should be cleared preventively. As much as these measures can be effective against fingerprinting, they also impose serious restrictions on web applications, limit overall usability, and decrease user experience. Fingerprinting detection can also be problematic and have questionable precision. In practice, it takes time to detect that a fingerprint is indeed being computed. As a page can immediately send the values being used for fingerprinting to the server, the page can learn a partial fingerprint before it is detected and blocked.

4 JShelter Design Decisions

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

JShelter does not aim on providing a perfect solution either. Our goals are as follow:

1. Create an webextension because webextensions work across multiple browsers and consequently can be easily installed into any browser that support webextension including Firefox and all browsers based on Chromium.
2. Do not create a perfect solution, rather focus on what other webextension lack: consistent approach to the threat T2 and a protection from T3, T4[12], T5, and T6.
3. Make the webextension friendly for people without technical knowledge.

Chrome Zero [35] and Web API Manager [49] 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 modifies or disables JavaScript APIs, it aims on threats T2, T3, and T6. (2) Fingerprint Detector provides heuristic analysis of fingerprinting behaviour and tackles T2

Fingerprint detector report, figure

A slightest mismatch between results of two APIs can make user more visible to fingerprinters [28, 38]. Hence, all protections are considered from the point of fingerprintability and the threat to leak information about the browser or user and other threats presented in Sect. 2. When it does not require to much work, JShelter tries to mimic a stationary device with consistent and plausible readings.

4.1 Fingerprinting protection

JShelter 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 in different way on different domains so that the cross-site fingerprint is not stable [**?**]. The focus is on applying security counter-measures 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 webistes and in each session.

In addition we offer a fingeprinting detector that monitors the protected APIs and other APIs that are commonly used by fingerprinters similarly to previous researchers [28**?**]. When a fingerprinting attempt is detected, the user is notified and optionally can block further HTTP requests by the page and delete all storage. The goal is to prevent computing the full fingerprint and uploading it to the server. Should the fingerprinter collect all values and compute the fingerprint before any upload, the aggressive mode completely prevents the fingerprinting. However, the fingerprinter can gradually upload detected values and a partial fingerprint can leak the browser.

4.2 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 but, for example, magnetometer readings have big fluctuations. Magnetometer fluctuation is simulated 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 that could negatively impact the browser's performance.

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

12 Currently, JShelter does not provide any protection for T4 but we plan to add such support in the future

4.3 Network Boundary Shield

Network Boundary Shield prevents web pages to use the browser as a proxy between local network and the public Internet. [**?**] for examples of attacks handled by the Network Boundary Shield. The protection encapsulates the WebRequest API, so it captures all outgoing requests.

4.4 Code ported from Chrome Zero

4.5 Farbling-like prevention of browser fingerprinting

We decided to borrow the protection implemented in Brave[13] and apply the same or very similar protection in JShelter. Farbling uses generated session and eTLD+1 keys to deterministically change outputs of APIs commonly used for browser fingerprinting. 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. But

2 iterations, significant performance hit

4.6 Fingerprint detector

We developed a JShelter module dedicated to browser fingerprinting detection called FingerPrint Detector (FPD). FPD applies a heuristic approach to detect fingerprinting behaviour in real-time. FPD counts calls to JavaScript APIs often employed by fingerprinting scripts. When FPD detects fingerprinting attempt, it will (1) inform the user, (2) prevent uploading of the fingerprint to the server, (3) prevent storing the fingerprint for later usage. The heuristic approach was chosen as many prior studies [2, 12] 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.

Nevertheless, we expect that the APIs for fingerprinting will change in time so that we designed the heuristics as flexible as possible. We expect to run periodic web crawls based on the tools initially developed by Snyder et al. [**?**]. 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.

Our heuristics are based on knowledge and rules that originated from previous studies. Iqbal et al. [24] 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 [12] to detect additional fingerprinting techniques. We looked through the source code of fingerprinting tools like FingerprintJS[14], Am I Unique[15] and Cover Your Tracks[16]. Furthermore, we analyzed existing detection tools, namely A Fingerprinting Monitor For Chrome (FPMON)[17] and Don't FingerPrint Me (DFPM)[18].

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. Once a fingerprint attempt is detected, JShelter informs 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 brake the page.

4.7 Early modification of JavaScript environment

The core functionality of the extension lays in modifying the results of the built-in JavaScript APIs and built-in object behaviour. JShelter employs the same mecha-

13 See https://github.com/brave/brave-browser/issues/8787 and https://github.com/brave/brave-browser/issues/11770

14 https://github.com/fingerprintjs

15 https://amiunique.org/

16 https://coveryourtracks.eff.org/

17 https://fpmon.github.io/fingerprinting-monitor/

18 https://github.com/freethenation/DFPM

nism proposed by Schwarz et al. [35] in Chrome Zero. However, Chrome Zero was a proof-of-concept without any modification in the last 4 years. As Shusterman et al. [46] note, there are several problems with the Chrome Zero: original implementation available through prototype chain, ...

Moreover, current webextension APIs lack a reliable way to inject scripts modifying the JavaScript environment before page scripts starts running and have the opportunity to store the original API calls. Firefox suffers from a long-standing unfixed bug 1267027[19] that possibly prevents 10 % of Firefox extensions from working correctly on pages with Content-Security-Policy prohibiting inline scripts [23]. A significant effort of the JShelter development went into developing a reliable cross-browser early script injection.

FIXME Giorgio, describe our approach, Chrome debug interface, Firefox shielding and the related problems — we need to make extension objects available for page scripts and vice-versa, this is costly, needs a lot of code for compound objects. iframes, nested iframes, etc. Possibly introduce NSCL

5 Results

5.1 Crawl study

We have developed web crawler[20] that collects JavaScript calls made by each visited website. The crawler is based on OpenWPM[21] and a modified web browser extension Web API Manager[22] that collects statistics of called JavaScript APIs.

The crawler visits the most visited website from the Tranco lists and observes the difference in calls without a privacy preserving extension like uBlock Origin. The goal is to identifiy the APIs that are called by the page without the privacy preserving extension and blocked by the extension. As the blocked content usually perform fingerprinting, analytics, or other activities that are not necessary, such data would train the fingerprint detector not only for fingerprinting but also for other blocked behaviour. It is an open research question if this is desired or not.

For now, we only visited the homepages because we wanted to visit as many different websites as possible. In the future, we plan to launch long-term crawling, which will include subpages. In particular, we want to focus on visiting login pages, where we expect fingerprint scripts to be prevalent. Then, we can compare API calls on login pages and other pages.

We tried to visit the first 250 000 websites from the Tranco list[23]). 211 843 homepages of websites from the Tranco list were successfully visited in both modes - with and without uBlock Origin. More than 4 000 000 000 JS calls were intercepted and stored into SQLite databases[24].

The crawl identified JavaScript endpoints, often used to create a browser fingerprint. The observed data allow assigning weights for each endpoint and future improvements of Fingerprint Detector.

5.2 FPD study

It is a great challenge to differentiate between benign and fingerprinting usage of a JavaScript APIs. Hence, the heuristics approach needs careful fine-tuning. We focused on targeting mainly excessive fingerprints and keeping a number of false positives as low as possible. As FPD strictly blocks all subsequent requests, we must ensure that this blocking occurs only in the necessary cases when there is a high likelihood of fingerprinting. 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[25]. Inaccessible websites were randomly replaced by ones from the top 200 list. For each page visit, we wiped browser settings to ensure determinism of initial access. As the erasure removed any previously-stored identifier, 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. To see an impact of a

19 https://bugzilla.mozilla.org/show_bug.cgi?id=1267027
20 https://github.com/martinbednar/web_crawler/
21 https://github.com/openwpm/OpenWPM
22 https://github.com/pes10k/web-api-manager

23 X79N, https://tranco-list.eu/list/X79N/1000000
24 https://nextcloud.fit.vutbr.cz/s/XKm3PCZnr2xkPH9
25 https://tranco-list.eu/list/23W9/1000000

	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 manual crawl of the top 100 web pages according to the Tranco list.

browser on the detection process, we used both Google Chrome[26] and Mozzila Firefox[27].

We needed the ground truth for web pages employing fingerprinting. We used both FPMON and DFPM extensions to create the ground truth. We selected these two extensions because they are the only ones capable of real-time fingerprinting detection. 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.

(1) *The score of 6.* FPD successfully detects such pages. The only exception was *Google login* page. FPD does not detect excessive fingerprinting there as other extensions did. *Google login* occurred six times in total during testing and we count them as false negatives. Nevertheless, the final fingerprint is not aggressive enough to provide enough entropy to identify most users uniquely.

(2) *The score of 4.* We still classify these web pages as deploying fingerprinting. FPD managed to detect all web pages with two exceptions, *Facebook login* page and `yandex.ru`. Both are border-line cases that do not obtain enough entropy.

(3) *The score of 3.* FPMON and DFPM treat browser fingerprinting differently, so we observed a few web pages with this score. It is questionable how to classify these pages when the reference extensions conflict. FPD detected fingerprinting only on one of these pages, *Paypal login* page. We consider this detection justified as we found clear tracks of canvas fingerprinting.

(4) *The score of 2.* We assume that web pages with this score may or may not be fingerprinting and the fingerprint is likely short on entropy. Moreover, these web pages are prone to misclassification because they may be close to the heuristic threshold. FPD detected two web pages with this score, namely *Cloudflare login* page and *Washington Post login* page. A closer analysis revealed that both pages use canvas fingerprinting in conjunction with other fingerprinting methods.

(5) *The score of 1 or 0.* FPD should not detect such web pages as fingerprinting. However, FPD detected fingerprinting on `ebay.com`. Manual inspection showed that `ebay.com` did indeed fingerprint using canvas fingerprinting, audio fingerprinting and other techniques.

In conclusion, we classify a page to be fingerprinting when its score is above or equal to 4. We did not count pages with the score of 3 or 2 as fingerprinting because they may not be engaged in fingerprinting in reality. It means that FPD may or may not detect such pages; we do not count such classification as an error in both cases. As discussed above, we manually inspected FPD in these situations. Finally, we consider anything below the score of 2 as not fingerprinting.

Study results are shown in table 1. Different heuristic thresholds of the extensions caused the main difference during testing. However, as we found out, the ground truth is far from being flawless. We encountered many exceptions during testing and examined them in detail. In many cases, FPD detects fingerprinting, but the reference extensions do not. For `ebay.com`, neither FPMON nor DPFM identified the ongoing fingerprinting. We got a very low false positive rate and an acceptable false negative rate in terms of methodology.

We also observed other notable behaviour during the testing. The asymmetry between detection on different browsers was minor and had minimal impact on detection. Moreover, FPD automatically recalculates heuristics to compensate for unsupported APIs. Finally, note that blocking tools like *adblockers* can significantly reduce the number of positive detections. These tools use filter lists to block tracking scripts before their execution. Using FPD with a filter-based blocking tool can significantly improve user experience and privacy.

5.3 Sensors

We discovered a loophole in the `Sensor.timestamp` attribute[28]. The value describes when the last `Sensor.onreading` event occurred, in millisecond pre-

26 https://www.google.com/chrome/

27 https://www.mozilla.org/en-US/firefox/

28 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 Xi-

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

Fig. 2. Magnetometer readings.

cision. We observed 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 exactly the same time.

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

Figure 2 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 Earch 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 environment. Fig. 2 (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 Feedback from users

JShelter users also reported increased number of false positives when using DNS-based filtering programs. If you use one, make sure that DNS returns 0.0.0.0 for the blocked domains.

5.5 eBay, block the page

Some web pages, like ebay.com, scan (some users) for opened local TCP ports to detect bots having opened 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. 3 for an example.

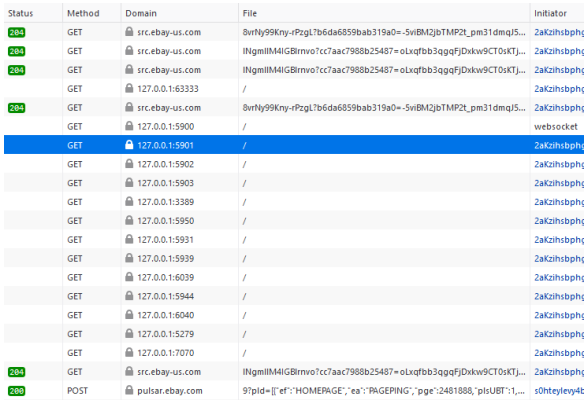

Fig. 3. eBay webpage scanning the local computer for open ports.

Although it could be that the underlying intentions are benign and users actually do benefit from the scanning, the scanning raises some ethical issues.

Very often, security and privacy are interconnected. But sometimes, one might increase security by revealing something private. In this case, the script information about the running device that is not obvious to the device owner (a user or a company). Typically, the owner of the device does not even know that such information can leak. If the information stays with ThreatMetrix, then the benefits could appear to be greater than the disadvantages. However, adversaries could stole information from ThreatMetrix. As the Article 29 Working Party clarified [**?** , use case 7.5], user-centric security

aomi Redmi Note 5; Android 9, kernel 4.4.156-perf+, Build/9 PKQ1.180901.001, Chrome 94.0.4606.71

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 the port scanning is a part of the login mechanism, open ports are personal data without doubts. So GDPR also applies. GDPR also list security as a possible legitimate interest of a data controller (e.g. eBay), see recital 49. Nevertheless, if such a scan is proportionate is an open question; it is possible that the legitimate interests of data controllers (such as eBay) are overriden by the interests or fundamental rights and freedoms of the data subject (you), see Article(6)(1)(f). The Court of Justice of EU (CJEU) decided several issues that concerned legitimate interests and the necessity of processing, e.g. C-13/16, C-708/18.

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[29] suggests that eBay and its processor ThreatMetrix are secretive about data being collected.

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

When we developed the Network Boundary Shield we did not anticipated localhost port scanning. When we first encountered the eBay port scanning case, we knew that this behaviour should trigger Network Boundary Shield. The tests revealed that Network Boundary Shield indeed works and our users are protected from this behaviour.

5.6 clockskew

6 Discussion and future work

[52]

JShelter should not be considered a single bullet proof solution

First level: NoScript+ublock+decentraleyes+FPD+NBS
Web API Manager?
Final level: API finetuning by JShelter
[38]: cloakX [45]

The research question concerning farbling measures is if the little changes are enough for a determined fingerprinter that can, for example, approximate color values of several pixels or repeat an effect multiple times.

FPD downside: gradually sent fingerprint, security reasons on login pages, strict thresholds, page breakage.

JShelter currently support only a device positioned on a flat surface. We consider improvements like a moving device for a future work.

Magnetometer: Nevertheless, the generator uses a series of constants, whose optimal values should be a subject of future research and improvements. Perhaps, a correlation analysis with real mesurements could help in the future.

7 Conclusion

8 Acknowledgement

NLNET, IGA

References

[1] 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/.

[2] 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.

[3] 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.

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

[5] 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-

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

network-en_0.pdf.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

[42] 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`.

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

[44] 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.

[45] 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.

[46] 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. URL `https://www.usenix.org/conference/usenixsecurity21/presentation/shusterman`.

[47] 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.

[48] 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`.

[49] 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`.

[50] 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.

[51] 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.

[52] Oleksii Starov and Nick Nikiforakis. Privacymeter: Designing and developing a privacy-preserving browser extension. In Mathias Payer, Awais Rashid, and Jose M. Such, editors, *Engineering Secure Software and Systems*, pages 77–95. Springer International Publishing, Cham, 2018. ISBN 978-3-319-94496-8.

[53] 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`.

[54] 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.

[55] 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. URL `http://doi.acm.org/10.1145/2810103.2813632`.

[56] 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.

[57] 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.