FP-TESTER: Automated Testing of Browser Fingerprint Resilience

Antoine Vastel, Walter Rudametkin, Romain Rouvoy

FP-TESTER: Automated Testing of Browser Fingerprint Resilience

Antoine Vastel
Univ. Lille / Inria

Walter Rudametkin
Univ. Lille / Inria

Romain Rouvoy
Univ. Lille / Inria / IUF

***Abstract*—Despite recent regulations and growing user awareness, undesired browser tracking is increasing. In addition to cookies, *browser fingerprinting* is a stateless technique that exploits a device's configuration for tracking purposes. In particular, browser fingerprinting builds on attributes made available from Javascript and HTTP headers to create a unique and stable fingerprint. For example, browser plugins have been heavily exploited by state-of-the-art *browser fingerprinters* as a rich source of entropy. However, as browser vendors abandon plugins in favor of extensions, fingerprinters will adapt.**

We present FP-TESTER, an approach to automatically test the effectiveness of browser fingerprinting countermeasure extensions. We implement a testing toolkit to be used by developers to reduce *browser fingerprintability*. While countermeasures aim to hinder tracking by changing or blocking attributes, they may easily introduce subtle side-effects that make browsers more identifiable, rendering the extensions counterproductive. FP-TESTER reports on the side-effects introduced by the countermeasure, as well as how they impact tracking duration from a fingerprinter's point-of-view. To the best of our knowledge, FP-TESTER is the first tool to assist developers in fighting browser fingerprinting and reducing the exposure of end-users to such privacy leaks.

I. INTRODUCTION

Browsers are critical applications for most end-users because they represent the primary vector to access the Internet and online web applications. However, their success encourages advertisers to continuously track user sessions for profiling purposes [1], [2], [3], [4]. While cookies remain the most widespread technique for tracking, researchers have brought to light *browser fingerprinting* as another, complementary technique. Contrary to cookies that store an identifier on the client's device, browser fingerprinting is a stateless technique that leverages a set of attributes accessible in the browser to generate a unique identifier. The combination of attributes is called a *fingerprint* and is mostly collected from Javascript or HTTP headers. The more entropy in attributes, the more accurate the fingerprints.

However, fingerprinting evolves due to the inclusion and removal of attributes, such as HTML5 Canvas or the deprecation of NPAPI plugins [5]. Recently, Starov *et al.* [6] showed that browser extensions may be used as a new fingerprinting vector. They detect the presence of extensions by observing side-effects on the DOM and showed that among $1,656$ extensions that introduce DOM changes, $90\,\%$ can be uniquely identified. This means that, paradoxically, even extensions implementing fingerprinting countermeasures could be revealed and thus exploited to track users.

In this paper, we present an approach to reduce the fingerprintability of browser fingerprinting countermeasures. Although we focus on countermeasure extensions, our approach can be used to evaluate the fingerprintability of any extension. More specifically, we propose FP-TESTER—a testing toolkit that characterizes the fingerprintability of an extension at two levels: *a) short-term fingerprintability—i.e.*, at time t, how different is the fingerprint from a browser with and without the extension installed; *b) long-term fingerprintability—i.e.*, how does having the extension installed impact the ability to track a browser. Uniqueness and stability make fingerprint tracking feasible; the two levels we test give us insight into both.

The remainder of this paper is organized as follows. First, we introduce the related work on browser fingerprinting, in particular on extension fingerprintability (cf. Section II). Then, we present FP-TESTER, our approach to evaluate browser extension fingerprintability (cf. Section III). Finally, we report on the preliminary experiments we conducted with FP-TESTER, before concluding (cf. Sections IV & V).

II. RELATED WORK

Browser fingerprinting consists in identifying user devices from attributes made available by browsers. Most of these attributes are collected using Javascript, HTTP headers, or Flash when available. In 2010, Eckersley [7] showed that $83.6\,\%$ of the fingerprints they collected were unique. This high rate of uniqueness was in part due to the list of fonts (and other attributes obtained through Flash) and plugins, which had high entropy. More recently, in 2016, Laperdrix *et al.* [5] confirmed Eckersley's results by showing that $90\,\%$ of desktop fingerprints, and $81\,\%$ of mobile fingerprints were unique. Nevertheless, they also showed that the source of entropy in fingerprints had evolved since 2010. On the one hand, the decline of Flash and the deprecation of NPAPI plugins led to a decrease in entropy. On the other hand, new attributes such as HTML 5 canvas [8] provided even higher entropy.

As Javascript APIs are added and deprecated, fingerprinting evolves accordingly, continuously identifying new sources of entropy to maintain uniqueness. Recently, Starov *et al.* [6] used the list of installed extensions as a new source of entropy. They introduced the notion of *browser extension fingerprintability—i.e.*, how installed extensions make browsers more vulnerable to fingerprinting. To measure the fingerprintability of extensions, they observe changes to the DOM and

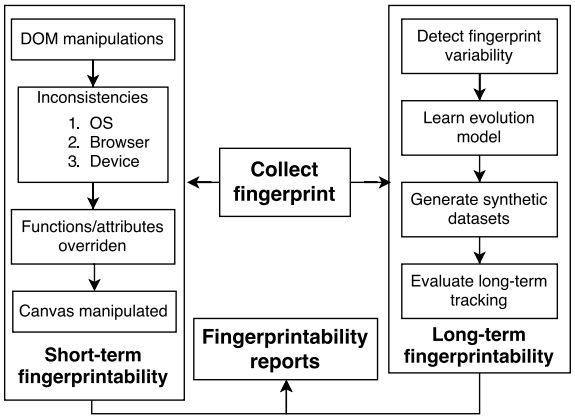

Fig. 1. Overview of FP-TESTER toolkit

extract a signature to identify the extensions. By analyzing $1,656$ extensions that perform detectable DOM changes, they showed that $90\,\%$ performed a unique combination of changes. Additionally, Acker *et al.* [9] showed that, for Google Chrome, it is possible to identify extensions using public resources. Besides methods to detect general purpose extensions, Nikiforakis *et al.* [10] demonstrated that it is possible to detect user-agent spoofer extensions—a specific kind of browser fingerprinting countermeasure—by looking at inconsistencies they introduce in a fingerprint. They analyzed 11 user-agent spoofing extensions and showed that they create unexpected combinations of attributes, called *inconsistencies*, making them detectable.

Based on these observations, we propose FP-TESTER, a toolkit to assist developers in reducing the fingerprintability of their browser extensions.

III. OVERVIEW OF FP-TESTER TOOLKIT

Figure 1 depicts an overview of FP-TESTER. We split the short and long term fingerprintability evaluations into two components based on the idea, expressed in previous research [11], [12], [13], that fingerprint tracking requires fingerprints to be both *unique* and *relatively stable over time*. In particular, Section III-A evaluates the *short-term fingerprintability* of an extension, which is related to uniqueness, while Section III-B studies how an extension influences *long-term fingerprintability* by measuring how uniqueness, randomness and instability affect tracking duration. More details on the architecture of FP-TESTER are given in Section III-E.

A. Short-Term Fingerprintability of Browser Extensions

Short-term fingerprintability reflects how, at a time t, the fingerprint generated by a browser with a countermeasure differs from a browser without it. We present 5 steps to evaluate short-term fingerprintability.

1) DOM modifications: We reuse the approach proposed by Starov *et al.* [6] to monitor DOM changes made by extensions. For example, ad blockers may remove ads from the DOM.

2) Inconsistencies: FP-TESTER also detects fingerprint inconsistencies. As shown by Nikiforakis *et al.* [10], countermeasures may introduce unexpected combinations of attributes, leading to *inconsistencies*. We propose a test suite to evaluate if the attributes of a fingerprint are globally consistent. These tests range from simple checks, such as verifying that the user agent in the HTTP headers and the `navigator` object are identical, to more subtle tests to verify that the rendering of an emoji in a canvas is consistent with the claimed OS [5]. We provide more details on these tests in Section IV.

3) HTTP headers modification: FP-TESTER scans HTTP headers to extract non-standard headers added by extensions.

4) Overridden functions/attributes: While there are different techniques to spoof a browser fingerprint, most extensions rely on overriding functions or getters involved in the fingerprinting process. Thus, we test if native JavaScript functions are overridden by looking at their `toString` representations.

5) Canvas fingerprinting: Finally, canvas fingerprinting has high entropy [5] and is central to countermeasures, such as CANVAS DEFENDER. Thus, FP-TESTER analyses canvas pixels to detect if they have been altered.

B. Long-Term Fingerprintability of Browser Extensions

This section focuses on how changes and side-effects introduced by countermeasures impact the ability to track a browser. Although being unique is required for tracking, and an extension may increase uniqueness, this is insufficient if the extension frequently changes the fingerprint. For example, users of an extension against canvas fingerprinting may have unique canvas attributes at any given time. However, if the extension continually randomizes the canvas, it may make tracking impossible. Although, if the extension has a very small user-base, none of this might matter, simply detecting it may increase fingerprintability. We present our methodology to evaluate long-term fingerprintability.

1) Detecting variability: FP-TESTER detects the variability introduced by an extension. For example, in the case of a simple user agent spoofer, only the user agent attribute varies, while a canvas extension may modify canvas pixels and override functions, such as *toDataURL*. To detect the changes an extension introduces, we collect fingerprints from the browser with and without the extension activated. Multiple fingerprints are needed if the extension changes or randomizes the fingerprints between executions. This allows FP-TESTER to learn how the extension alters the browser fingerprint. Because extensions may be highly configurable, different settings may influence the fingerprint. Developers can specify a profile parameter, which allows FP-TESTER to test and learn the variability model for each configuration.

2) Evaluating long-term tracking: Once FP-TESTER learns the extension's variability models, it evaluates how the extension exposes browsers to long-term tracking. Using genuine fingerprints obtained from the AMIUNIQUE database [5], FP-TESTER simulates the presence of the countermeasure in browser fingerprints and quantifies the effect the countermeasure would have on tracking algorithms. Simulating the

extension's effects on genuine fingerprints allows testing a wide range of configurations, without the overhead and difficulty of testing thousands of individual devices and browsers. To do so, FP-TESTER creates a population P composed of N genuine fingerprints, without inconsistencies, chosen randomly. This is necessary because most browsers do not use countermeasures, thus we attempt to recreate a realistic population to compare with. We randomly split P into 2 subpopulations, P_e and P_n, corresponding to the population of browsers with and without the extension, with a ratio f_e and f_n of N, respectively. FP-TESTER applies changes to all fingerprints in P_e so that they appear to originate from browsers with the countermeasure (detailed in Section III-C). Using all fingerprints in P, FP-TESTER executes the 2 linking algorithms from FP-STALKER [13], as well as a new algorithm that leverages inconsistencies revealed by a browser to target it more efficiently. This allows comparing the vulnerability to tracking of browsers from P_n (without the extension), to browsers from P_e (with the extension).

C. Applying Browser Extension Changes on Fingerprints

FP-TESTER simulates the effects of a countermeasure extension on genuine fingerprints. This process is complex and we do not claim to perfectly simulate the countermeasures since it may be difficult to recreate all their effects. However, even with imperfections, the effects are similar from the point-of-view of the tracking algorithms. As proposed in [13], in order to decide if two fingerprints originate from the same browser, the algorithms generate a feature vector that is mostly pairwise comparisons of attributes between each fingerprint. For example, in the case of a canvas countermeasure, FP-TESTER may have learned that `toDataURL` is always overridden, and that the canvas is randomly modified. Without generating a canvas that reproduces the precise effects of the countermeasure, FP-TESTER is capable of generating a fingerprint that simulates the use of the countermeasure and, when compared to another fingerprint, results in a feature vector that is equivalent to one obtained if we had installed the countermeasure in the browser. Since, the feature vector used by the algorithms tests for the equality of the canvas attributes between two fingerprints, we assign a random value to the canvas, as it only needs to be different from the previous values, and we keep the fact that `toDataURL` is overridden. For attributes that are split into sub-attributes, with particular semantics, special attention is needed to reproduce the countermeasure's effects. It is not sufficient to simply detect that the attribute changed since the feature vector relies on more fine-grained details. FP-TESTER must learn how to apply changes to these attributes. For example, the user agent reflects the browser, its version, and the operating system. In this case, a change may be a random substitution from a list of known user agents, or a more subtle change in the browser version, or something in between. For such attributes, FP-TESTER discriminates different types of changes to different sub-attributes and attempts to reproduce these changes in the simulated fingerprints so that the comparisons remain similar from the algorithm's point-of-view. Furthermore, the choice to apply a change or not at a given time depends on the frequency determined during the learning phase of the variability models. Some extensions change attributes very frequently, while others only once.

D. Sharing Fingerprintability Reports with Developers

For each of FP-TESTER's components, we deliver some actionable feedback to extension developers. Regarding short-term fingerprintability, we indicate: *1)* the modifications applied to the DOM, *2)* the list of inconsistencies, *3)* the functions overridden and how to hide them, and *4)* if FP-TESTER detected a canvas manipulation.

Regarding long-term fingerprintability, we provide projections of how tracking algorithms would perform on the normal population versus the population that emulates browsers with the countermeasure. We also test if differences exist in terms of tracking duration depending on the extension profile.

E. Implementation of FP-TESTER

We describe FP-TESTER's architecture, which is split into 4 components to make it modular and reusable.

a) Fingerprinter component: FP-TESTER provides a fingerprinting script that allows to easily include new attributes.

b) Short-term component: is composed of sub-components that all return `ShortTermResult` objects. This standardizes the sub-component test results and contains information such as name, description, category (*e.g.*, overridden, inconsistency), as well as whether the test succeeded and a message to explain what it implies. The modular architecture enables adding sub-components. The outputs are automatically integrated into the short-term fingerprintability report. One such sub-component is for emoji verification and uses supervised machine learning to verify that a rendered emoji is consistent with the OS. We do not directly map the raw values of an image with an emoji to a specific OS as, even for different devices on the same OS and browser, the size of the emoji and its position on the image differs. Moreover, raw image values would not work with canvas countermeasures. Regarding machine learning, a natural solution is to render an emoji in a canvas, extract the image, and train a classifier on top of it. However, we do not have enough emoji data to train a robust classifier. Instead, we trained our model on a subpart of the canvas included in the AMIUNIQUE dataset, which contains an emoji. Since the dataset contains more than 3 years of data, it is suitable to train a robust classifier. The classifier is based on a *convolutional neural network* given as an example to classify images from the CIFAR10 dataset, which we retrained on our dataset. Concerning inconsistencies, since they play a central role in our approach, we provide a mechanism to add new consistency rules. We define a consistency rule as tuple of 3 attributes `(att1, att2, link)` where `att1` and `att2` are the two attributes which have a consistency link, and `link` is a function that takes the two attributes as parameters, and returns `true` if they are consistent. As an example, the user agent can be obtained from the

`navigator` object and the HTTP headers, the tuple is defined as `(uaHttp, uaNavigator, eq)` where `eq` is the function that determines if the attributes are equal.

c) Long-term component: is composed of sub-components responsible for tasks such as the detection of the variability introduced by a countermeasure. Contrary to the short-term component, we do not support adding new sub-components to it. Nevertheless, it is still possible to extend it by adding new tracking algorithms.

d) Report component: is responsible for the generation of developer reports. Since values returned by short and long-term components are standardized, they can automatically be integrated into the report. Moreover, this component can be extended to add new visualizations.

IV. Experimentation & Preliminary Results

We evaluate the short-term fingerprintability component. We focus on the Random Agent Spoofer (RAS) for Firefox and detail the actionable feedback from FP-Tester. RAS is an extension that aims at protecting against browser fingerprinting. Although it is not available as a web extension—*i.e.*, not available on Firefox versions > 57—we use it as an example since it modifies many attributes commonly used in fingerprinting algorithms. RAS changes the values of attributes, such as the `user agent`, the `platform`, the list of `languages`, and `screen resolution`. To introduce fewer inconsistencies, RAS relies on a system of profiles—*i.e.*, combinations of attributes extracted from real browsers. It also allows disabling specific APIs used by fingerprinting, such as WebGL or WebRTC.

A. Collecting Browser Fingerprints

The first step of our evaluation is to collect fingerprints from browsers with RAS installed. To do so, we created a webpage that collects a fingerprint and stores it in a database. We put ourselves in the developer's place and collect fingerprints from a single computer used for development. In total, we collected 72 fingerprints, half from Linux, and the other half from two Windows virtual machines on the same computer. At each visit, we change the profile used by RAS.

B. Short-Term Fingerprintability Report

After the collect phase, FP-Tester looks for DOM manipulations, inconsistencies, overridden functions, and manipulated canvas pixels. Figure 2 presents how FP-Tester reports to the developer on the results of the analysis. Each node represents an attribute. A red edge between two nodes means that an inconsistency is detected between them. If a node's border is dashed, it means that FP-Tester detects that the attribute has been overridden. In the case of RAS, FP-Tester did not detect any DOM or canvas pixel manipulations, nor any non-standard HTTP headers, hence they are not in the image.

1) Inconsistencies: Although RAS uses a system of profiles, FP-Tester detected inconsistent combinations of attributes. Indeed, while attributes included in the profiles are consistent among themselves, it is not necessarily the case of other attributes, such as Javascript errors. For each collected fingerprint, FP-Tester extracts the browser, the OS and the device contained in the user agent, and uses them as references to detect inconsistencies. Thus, FP-Tester's report is divided into three groups corresponding to the browser, OS, and device-related attributes. For each group, we describe one of the tests that failed.

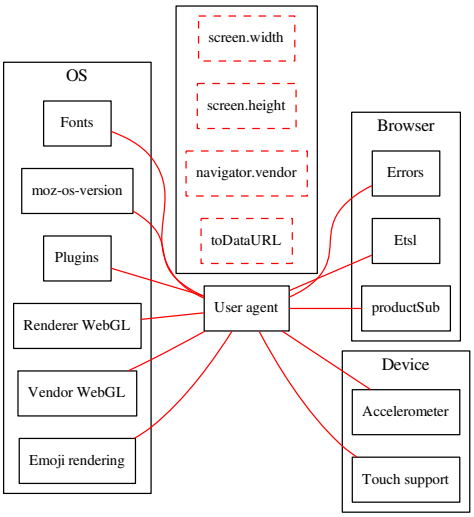

Fig. 2. Short-term fingerprintability report provided by FP-Tester

a) OS inconsistencies: FP-Tester identified 6 tests that reveal OS inconsistencies. One corresponds to the way emojis are rendered, which depends on the OS [5]. FP-Tester uses a neural network trained on the AmIUnique dataset to predict from which OS the rendered emoji originates. Because RAS can change the OS in the user agent, it may fail this test.

b) Browser inconsistencies: RAS failed 3 tests to detect browser inconsistencies. We focus on etsl, which stands for `eval.toString().length`. Indeed, `eval.toString()` returns a string whose length depends on the browser. While Safari and Firefox display the same string with a length of 37 characters, the length is 33 on Chrome and 39 on Internet Explorer. FP-Tester was capable of detecting that the browser displayed in the user agent was sometimes modified.

c) Device inconsistencies: RAS failed 2 tests that reveal that it lied about the real nature of the device. In particular, some fingerprints claimed to be mobile devices or tablets, but they did not support mobile-specific events, such as `touchSupport` or sensors like the accelerometer.

2) Overridden Functions: FP-Tester analyzed the `navigator` and `screen` prototypes and detected that the getters of `navigator.vendor` and `screen.width/height` were overridden. It also detected that `toDataURL`, which is used to obtain a canvas value, was also overridden.

C. Other Countermeasures

We also evaluate the short-term fingerprintability of four more countermeasures: two canvas countermeasures, Canvas Defender and Canvas Fingerprint Block, as well as two user agent spoofers, Ultimate User Agent Switcher

(UUAS) and USER-AGENT SWITCHER (UAS). FP-TESTER detected inconsistencies introduced by all four extensions. Regarding canvas countermeasures, it spotted inconsistencies at different levels: overridden functions (`toDataURL`), as well as the canvas pixels. Concerning the user agent spoofers, UUAS is detected because it alters the user agent sent in the HTTP requests, but doesn't alter the one in the `navigator` object. Although UAS changes both user agents, it does not modify other attributes, such as `navigator. platform`, which should be consistent with the OS.

D. Identification of countermeasures

Beyond detecting the existence of a countermeasure, identifying the countermeasure could increase entropy. Of the five countermeasures we tested, we were able to identify RAS, as well as the two canvas extensions. We failed to reliably identify the two user agent spoofers solely by looking at the fingerprint or the side effects they introduce—*i.e.*, without using the technique presented in the related work that relies on Chrome public resources [9]. Concerning the two canvas countermeasures, they can be identified by looking at the string representation of `toDataURL` since, in both cases, it leaks the code used by the extension to modify the native behavior. For RAS, no single attribute can be used to identify the extension. Instead, we test for the presence of side-effects introduced by RAS which, when combined, can reveal a signature of its presence. For example, in Figure 2, one can observe that RAS overrides `screen.width/height`, `navigator.vendor` and `toDataURL`. Thus, while we do not have a single attribute to identify RAS, we can still use the combination of multiple attributes as a signature.

E. Discussion

We showed that it is possible to detect the presence of countermeasures, and for some of them to reveal their identity. Nevertheless, being detected with a countermeasure, or being unique, does not necessarily imply being tracked more easily, hence the distinction between short and long-term fingerprintability. We argue that the effect an extension may have on tracking depends on different factors, such as being able to identify the countermeasure, the number of users of the countermeasure, the ability to recover the real fingerprint values, and the volume of information leaked by the countermeasure. Thus, while it might not always be possible to provide an effective and undetectable countermeasure, it is important for countermeasure developers to evaluate if the anonymity gained from the countermeasure outweighs the information the countermeasure leaks. We believe FP-TESTER can help developers answer this question.

F. Threats to Validity

One of the limits of FP-TESTER in its current version is its usage of rules to evaluate the short-term fingerprintability of a countermeasure. Indeed, our ruleset is not exhaustive and may miss some privacy leaks. It also needs to be updated to take into account new browser features that could be used for fingerprinting. Nevertheless, as we explain in Section III-E, FP-TESTER aims to be modular in order to easily add new rules. Moreover, although this paper reports on rules we learned manually, we are currently working on automating the inference of such rules to facilitate the maintenance of the ruleset. Another threat lies in the way countermeasure developers interpret the report generated by FP-TESTER. As we explain in the discussion, being detected does not necessarily make tracking easier. It is important to present the information in a way that does not mislead the developer. For example, if we consider the case of a canvas countermeasure, the entropy from the identification of the countermeasure may be compensated by the decrease of entropy from the removal of the canvas.

V. CONCLUSION

In this paper, we introduce FP-TESTER, our approach to support developers in improving the resilience of their countermeasures against fingerprinting. Our work starts from techniques proposed by Starov *et al.* [6] to detect changes made to the DOM. Besides DOM monitoring, FP-TESTER also looks for inconsistencies, overridden native Javascript functions, and manipulated canvas pixels. While short-term fingerprintability—how different is the fingerprint with the extension installed—is an important criterion for tracking, for long-term fingerprintability, FP-TESTER also uses a strategy that simulates the effects of countermeasures on genuine fingerprints and tests if the browsers with the countermeasure can be tracked more easily or not. Finally, we provide a fingerprintability report to extension developers on both short and long-term exposure to fingerprinting, allowing them to take appropriate actions to reduce their vulnerability to tracking.

REFERENCES

[1] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016," in *Usenix Security*, 2016.

[2] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," in *CCS*, 2016.

[3] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *CCS*, 2014.

[4] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "Fpdetective: dusting the web for fingerprinters," in *CCS*, 2013.

[5] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints," in *S&P*, 2016.

[6] O. Starov and N. Nikiforakis, "XHOUND: Quantifying the Fingerprintability of Browser Extensions," in *S&P*, 2017.

[7] P. Eckersley, "How unique is your web browser?" in *PETS*, 2010.

[8] K. Mowery and H. Shacham, "Pixel Perfect: Fingerprinting Canvas in HTML5," in *W2SP*, 2012.

[9] S. V. Acker and A. Sabelfeld, "Discovering Browser Extensions via Web Accessible Resources," in *CODASPY*, 2017.

[10] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *S&P*, 2013.

[11] P. Laperdrix, W. Rudametkin, and B. Baudry, "Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification," in *SEAMS*, 2015.

[12] N. Nikiforakis, W. Joosen, and B. Livshits, "PriVaricator: Deceiving Fingerprinters with Little White Lies," in *WWW*, 2015.

[13] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "FP-STALKER: Tracking Browser Fingerprint Evolutions," in *S&P*, 2018.