A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication

Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, Alexandre Garel

A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication

NAMPOINA ANDRIAMILANTO, Institute of Research and Technology b<>com, France and Univ Rennes, CNRS, IRISA, France
TRISTAN ALLARD, Univ Rennes, CNRS, IRISA, France
GAËTAN LE GUELVOUIT, Institute of Research and Technology b<>com, France
ALEXANDRE GAREL*, Institute of Research and Technology b<>com, France

Modern browsers give access to several attributes that can be collected to form a browser fingerprint. Although browser fingerprints have primarily been studied as a web tracking tool, they can contribute to improve the current state of web security by augmenting web authentication mechanisms. In this paper, we investigate the adequacy of browser fingerprints for web authentication. We make the link between the digital fingerprints that distinguish browsers, and the biological fingerprints that distinguish Humans, to evaluate browser fingerprints according to properties inspired by biometric authentication factors. These properties include their distinctiveness, their stability through time, their collection time, their size, and the accuracy of a simple verification mechanism. We assess these properties on a large-scale dataset of 4, 145, 408 fingerprints composed of 216 attributes and collected from 1, 989, 365 browsers. We show that, by time-partitioning our dataset, more than 81.3% of our fingerprints are shared by a single browser. Although browser fingerprints are known to evolve, an average of 91% of the attributes of our fingerprints stay identical between two observations, even when separated by nearly 6 months. About their performance, we show that our fingerprints weigh a dozen of kilobytes and take a few seconds to collect. Finally, by processing a simple verification mechanism, we show that it achieves an equal error rate of 0.61%. We enrich our results with the analysis of the correlation between the attributes and their contribution to the evaluated properties. We conclude that our browser fingerprints carry the promise to strengthen web authentication mechanisms.

CCS Concepts: • **Security and privacy** → **Multi-factor authentication**; • **Information systems** → *Browsers*.

Additional Key Words and Phrases: browser fingerprinting, web authentication, multi-factor authentication

ACM Reference Format:
Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. 2021. A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication. *ACM Trans. Web* 1, 1 (October 2021), 63 pages. https://doi.org/10.1145/3478026

1 INTRODUCTION

Web authentication widely relies on the use of identifier-password pairs defined by the end user. The password authentication factor is easy to use and to deploy, but has been shown to suffer from severe security flaws when used without any additional factor. Real-life users indeed use

This paper is a major extension (more than 50% of new material) of work originally presented in [16].
*The author participated to the fingerprint collection and analysis when working at the institution, but is not anymore affiliated to it.

Authors' addresses: Nampoina Andriamilanto, nampoina.andriamilanto@b-com.com, Institute of Research and Technology b<>com, 1219 avenue Champs Blancs, Cesson-Sévigné, France, 35510 and Univ Rennes, CNRS, IRISA, 263 avenue du général Leclerc, Rennes, France, 35000; Tristan Allard, tristan.allard@irisa.fr, Univ Rennes, CNRS, IRISA, 263 avenue du général Leclerc, Rennes, France, 35000; Gaëtan Le Guelvouit, gaetan.leguelvouit@b-com.com, Institute of Research and Technology b<>com, 1219 avenue Champs Blancs, Cesson-Sévigné, France, 35510; Alexandre Garel, alexandre.garel@b-com.com, Institute of Research and Technology b<>com, 1219 avenue Champs Blancs, Cesson-Sévigné, France, 35510.

2021. 1559-1131/2021/10-ART $15.00
https://doi.org/10.1145/3478026

common passwords [47], which paves the way to brute-force or guessing attacks [25]. Moreover, they tend to use similar passwords across different websites [31], which increases the impact of successful attacks. Phishing attacks are also a major threat to the use of passwords. Over the course of a year, Thomas et al. [96] achieved to retrieve 12.4 million credentials stolen by phishing kits. These flaws bring the need for supplementary security layers, primarily through multi-factor authentication [26], such that each additional factor provides an *additional security barrier*. However, this usually comes at the cost of *usability* (i.e., users have to remember, possess, or do something) and *deployability* (i.e., implementers have to deploy dedicated hardware or software, teach users how to use them, and maintain the deployed solution).

In the meantime, *browser fingerprinting* [58] gains more and more attention. The seminal Panopticlick study [33] is the first work to highlight the possibility to build a *browser fingerprint* by collecting attributes from a browser (e.g., the `userAgent` property of the `navigator` JavaScript object). In addition to being widely used for web tracking purposes [9, 34, 49] (raising legal, ethical, and technical issues), browser fingerprinting is already used as an additional web authentication factor *in real-life*. The browser fingerprints constitute a supplementary factor that is verified at login with the other factors, as depicted in Figure 1 (see Section 6.1 for an example of an authentication mechanism that relies on browser fingerprints). Browser fingerprints are indeed a good *candidate* as an additional web authentication factor thanks to their distinctive power, their frictionless deployment (e.g., no additional software or hardware to install), and their usability (no secret to remember, no additional object to possess, and no supplementary action to carry out). As a result, companies like MicroFocus [37] or SecureAuth [85] include this technique in their authentication mechanisms.

However, to the best of our knowledge, no large-scale study rigorously evaluates the adequacy of browser fingerprints as an additional web authentication factor. On the one hand, most works about the use of browser fingerprints for authentication concentrate on the design of the authentication mechanism [41, 56, 62, 76, 80, 88, 99]. On the other hand, the large-scale empirical studies on browser fingerprints focus on their effectiveness as a web tracking tool [33, 44, 60, 77]. Such a mismatch between the understanding of browser fingerprints for authentication – currently poor – and their ongoing adoption in real-life is a serious harm to the security of web users. The lack of documentation from the existing authentication tools (e.g., about the used attributes, about the distinctiveness and the stability of the resulting fingerprints) only adds up to the current state of ignorance, all this whereas security-by-obscurity directly contradicts the most fundamental security principles. Moreover, the distinctiveness of browser fingerprints that can be achieved when considering a wide-surface of fingerprinting attributes on a large population is, to the best of our knowledge, unknown. On the one hand, the studies that analyze browser fingerprints in a large-scale (more than $100,000$ fingerprints) consider fewer than 35 attributes [33, 44, 60, 62, 101]. This underestimates the distinctiveness of the fingerprints (e.g., [44] and [62] report a unique fingerprint rate of about 35%), as it increases the chances for browsers to share the same fingerprint. All this whereas more than a hundred attributes are accessible. On the other hand, the studies that consider more than fifty attributes either work on less than two thousands users [54, 77], or do not analyze the resulting fingerprints at all [11]. The current knowledge about the hundreds of accessible attributes (e.g., their stability, their collection time, their correlation) is, to the best of our knowledge, also incomplete. Indeed, previous studies either consider few attributes [28, 33, 35, 44, 60, 71, 78, 86] or focus on a single aspect of them (e.g., their evolution [62, 101]).

Our contributions. We conduct the first *large-scale data-centric empirical study* of the *fundamental properties of browser fingerprints* when used as an additional web authentication factor. We base our findings on an in-depth analysis of a real-life fingerprint dataset collected over a period of 6 months, that contains $4,145,408$ fingerprints composed of 216 attributes. In particular, our dataset

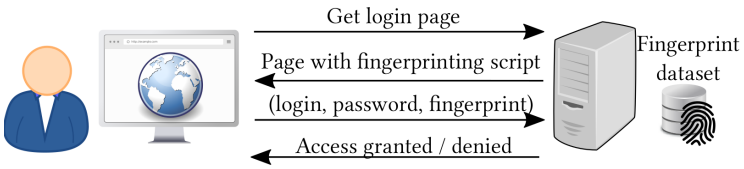

Fig. 1. A simplified web authentication mechanism that relies on browser fingerprinting.

includes nine *dynamic attributes* of three types, which values depend on instructions provided by the fingerprinter: five HTML5 canvases [28], three audio fingerprinting methods [78], and a WebGL canvas [71]. The dynamic attributes are used within state-of-the-art web authentication mechanisms to mitigate replay attacks [56, 80]. Each dynamic attribute has been studied singularly, but their fundamental properties have not yet been studied simultaneously on the same browser population. To the best of our knowledge, no related work considers a dataset of this scale, in terms of both fingerprints and attributes, together with various dynamic attributes. We formalize, and assess on our dataset, the properties necessary for paving the way to elaborate browser fingerprinting authentication mechanisms. We make the link between the digital fingerprints that distinguish browsers, and the biological fingerprints that distinguish Humans, to evaluate browser fingerprints according to properties inspired by biometric authentication factors [39, 66, 107]. The properties aim at characterizing the adequacy and the practicability of browser fingerprints, independently of their use within future authentication mechanisms. In particular, we measure the size of the browser anonymity sets through time, the proportion of identical attributes between two observations of the fingerprint of a browser, the collection time of the fingerprints, their size, the loss of efficacy between device types, and the accuracy of a simple illustrative verification mechanism. To comprehend the results obtained on the complete fingerprints, we include an in-depth study of the contribution of the attributes to the properties of the fingerprints. Moreover, we discuss the correlation between the attributes, make a focus on the contribution of the dynamic attributes, and provide the exhaustive list of the attributes together with their properties. To the best of our knowledge, no previous work analyzed browser fingerprinting attributes at this scale, in terms of the number of attributes, the number of fingerprints, and the variety of properties (e.g., stability, collection time).

In a nutshell, we make the following contributions:

(1) We formalize the fundamental properties that browser fingerprints should provide to be usable and practical as a web authentication factor.
(2) About their adequacy, we show that (1) considering a wide surface of 216 fingerprinting attributes on our large population provides a proportion of unique fingerprints – also called unicity rate – of 81.8% on our complete dataset, (2) by time-partitioning our dataset, the unicity rates are stable on the long term at around 81.3%, and 94.7% of our fingerprints are shared by 8 browsers or fewer, (3) on average, a fingerprint has more than 91% of identical attributes between two observations, even when separated by nearly 6 months, (4) our mobile browsers lack distinctiveness, as they show a unicity rate of 42%.
(3) About their practicability, we show that (1) the generated fingerprints weigh a dozen of kilobytes, (2) they are collected within seconds, (3) the accuracy of a simple illustrative verification mechanism is close to perfect, as it achieves an equal error rate of 0.61%.
(4) We enrich our results with (1) a precise analysis of the contribution of each attribute (a) to the distinctiveness, and show that 10% of the attributes provide a normalized entropy higher than 0.25, (b) to the stability, and show that 85% of the attributes stay identical for 99% of the

consecutive fingerprints coming from the same browser, (c) to the collection time, and show that only 33 attributes take more than 5ms to collect, (d) to the fingerprint size, and show that only 20 attributes weigh more than 100 bytes, (2) a discussion about the correlation of the attributes, and show that only 49 attributes can completely be inferred when knowing another attribute, (3) a focus on the properties of the nine dynamic attributes.

(5) We provide an in-depth description of our methodology and our dataset, with the goal of making our results reproducible. In particular, we include the exhaustive list of the collected attributes together with their properties, and a detailed description of the preprocessing of the fingerprints.

This paper is a major extension of work originally presented in [16], and brings more than 50% of new material. Specifically, we clarify the obtained results by discussing them further, we evaluate the accuracy of a simple illustrative verification mechanism, we highlight the contribution of the attributes to the properties, and we provide a comprehensive list of the attributes, together with their properties and their concrete implementation. In a summary, we make the following additions:

(1) Section 2 gains a description of the studied browser population. This includes the share of the families of browser and operating system, and insights about the bias towards French browsers. We also add a description of the data preprocessing step, and a comparison between our dataset and the dataset of previous studies.
(2) Section 3 gains the accuracy of a simple illustrative verification mechanism as an additional performance property.
(3) Section 4 gets the results further discussed, and gains the results of the accuracy of the simple illustrative verification mechanism.
(4) Section 5 is entirely new. It discusses the contribution of the attributes to the fingerprint properties, the correlation between the attributes, and the properties of the dynamic attributes.
(5) Section 6 is entirely new. It describes how browser fingerprints can be integrated in a web authentication mechanism and discusses the attacks that are possible on such mechanism.
(6) Section 7 is entirely new. It provides related works about the use of browser fingerprinting for authentication.
(7) The appendices are entirely new. Appendix A describes the concrete implementation of the studied attributes for reproducibility. Appendix C provides the keywords used to classify the fingerprints. Appendix B discusses a side effect encountered by our fingerprinting script that results in anomalous high collection time of some fingerprints or attributes. Appendix D discusses a more complex verification mechanism that relies on distance functions on the attributes to compare fingerprints. Appendix E provides the complete list of the attributes with their properties (e.g., number of distinct values, stability).

The rest of the paper is organized as follows. Section 2 describes the dataset analyzed in this study. Section 3 presents and formalizes the properties evaluated in the analysis. Section 4 presents the experimental results. Section 5 breaks down the analysis to the attributes to comprehend the results on the complete fingerprints. Section 6 describes how browser fingerprinting can contribute to an authentication mechanism and discusses the attacks that are possible on such mechanism. Section 7 positions this study with the related works. Finally, Section 8 concludes.

2 DATASET

In this section, we describe the browser fingerprint dataset that is analyzed in this study. First, we present the conditions of the collection and describe the precautions taken to protect the privacy of the experimenters. Then, we detail the preprocessing steps to cleanse the raw dataset. Finally, we describe the working dataset and compare it with the large-scale datasets of previous studies.

2.1 Fingerprints collection

To study the properties of browser fingerprints on a real-world browser population, we launched an experiment in collaboration with the authors of the Hiding in the Crowd study [44], together with an industrial partner that controls one of the top 15 French websites according to the site ranking service Alexa [14]. The authors of the Hiding in the Crowd study only consider the 17 attributes of their previous work [60] and focus on the issue of web tracking. On the contrary, we consider in this work more than one order of magnitude more attributes – 216 attributes – and focus on the use of browser fingerprinting as an additional web authentication factor. Our dataset contains more fingerprints than [44] because two browsers that show different fingerprints for a given set of attributes can come to the same fingerprint if a subset of these attributes is considered. As a result, the authors of [44] remove more duplicated fingerprints due to the higher chances for a browser to present the same fingerprint for 17 attributes than for 216 attributes.

2.1.1 Experiment. The experiment consisted in integrating a fingerprinting script on two general audience web pages that are controlled by our industrial partner, which subjects are political news and weather forecast. The script was active between December 7, 2016, and June 7, 2017. It fingerprinted the visitors who consented to the use of cookies in compliance with the European directives 2002/58/CE [1] and 2009/136/CE [2]. To differentiate two browsers in future analysis, we assigned them a unique identifier (UID) as a 6-months cookie, which was sent alongside fingerprints. Similarly to previous studies [33, 60], we coped with the issue of cookie deletion by storing a one-way hash of the IP address computed by a secure cryptographic hash function. We refer the interested reader to Section 2.3 for more details on the measures taken to protect the privacy of the experimenters.

2.1.2 Browser fingerprinting attributes. The fingerprinting script used in the experiment includes 216 attributes divided in 200 JavaScript properties, together with their collection time, and 16 HTTP header fields. In particular, they include three types of dynamic attributes that comprise five HTML5 canvases [28], three audio fingerprinting methods [78], and a WebGL canvas [71].

We sought to evaluate the properties of browser fingerprints when considering as many attributes as possible, to estimate more precisely what can really be achieved. We compiled the attributes from previous studies and open source projects. If the value of an attribute is not accessible, a flag explaining the reason is stored instead, as it is still exploitable information. Indeed, two browsers can be distinguished if they behave differently on the inaccessibility of an attribute (e.g., returning the value `undefined` is different from throwing an exception). We also configure a timeout after which the fingerprint is sent without waiting for every attribute to be collected. The attributes that were not collected are set to a specific flag. The complete list of attributes and their properties is available in Appendix E.

Client-side attributes only consist of JavaScript properties. No plugins (e.g., Flash, Silverlight) are used due to their removal and replacement by HTML5 functionalities [94]. Moreover, the fingerprinting script collects the HTTP headers from requests sent by JavaScript, hence the dataset contains no fingerprint of browsers having JavaScript disabled.

2.2 Browser population bias

The previously presented datasets were collected through dedicated websites and are biased towards privacy-aware and technically-skilled persons [33, 60, 77]. Our dataset is more general audience oriented and is not biased towards this type of population. Nevertheless, the website audience is mainly French-speaking users, resulting in biases that we discuss below. We emphasize that the obtained browser fingerprints – and accordingly, their properties – depend on the browser

Table 1. Comparison of the share of browser and operating system families between the studies Panopticlick (PTC), AmIUnique (AIU), Hiding in the Crowd (HitC), the average share of each family computed by StatCounter for the worldwide and French browser populations between January 2017 and June 2017, and this study. The symbol - denotes missing information. The families are ordered from the most common to the least common in this study. The shares for the AIU and the HitC studies both come from the HitC paper in which they are provided for the desktop and the mobile browsers. We calculated and provide here the shares for the complete browser population for comparability. We stress that the classification methodology can vary between the populations that are compared, and refer to Appendix C for a description of our methodology.

	PTC [33]	AIU [60]	HitC [44]	Worldwide [90, 91]	France [3, 4]	**This study**
Starting date	01/2010	11/2014	12/2016	01/2017	01/2017	**12/2016**
Duration	3 weeks	3-4 months	6 months	6 months	6 months	**6 months**
Firefox	0.568	0.437	-	0.064	0.193	**0.268**
Chrome	0.142	0.398	-	0.531	0.478	**0.265**
Internet Explorer	0.126	0.043	-	0.046	0.065	**0.260**
Edge	-	-	-	0.017	0.040	**0.078**
Safari	0.077	0.079	-	0.144	0.193	**0.064**
Samsung Internet	-	-	-	0.033	0.029	**0.048**
Others	0.088	0.042	-	0.165	0.002	**0.017**
Windows-based	-	0.568	0.831	0.375	0.523	**0.821**
Windows 10	-	-	-	-	-	**0.338**
Windows 7	-	-	-	-	-	**0.312**
Other Windows	-	-	-	-	-	**0.171**
Android	-	0.061	0.087	0.385	0.189	**0.091**
Mac OS X	-	0.133	0.048	0.051	0.120	**0.051**
iOS	-	0.047	0.023	0.132	0.138	**0.026**
Linux-based	-	0.150	0.008	0.008	0.015	**0.008**
Others	-	0.041	0.003	0.049	0.015	**0.003**

population. A verifier that seeks to use browser fingerprints as an authentication factor should analyze the fingerprints of a browser population that is as close as possible to the target population. Some browser populations can show inadequate properties, like the standardized browsers of a university which lack distinctiveness [17]. In the following analysis (see Section 4), we evaluate our properties on the complete browser population and on the subpopulations of the browsers running on desktop and mobile devices. A finer-grained analysis – down to a given browser family in a given version – can be done by sampling the population on the wanted subset. Studying the fingerprints at this level is out of the scope of this paper. We let such analysis as future works.

We match the `userAgent` JavaScript property with manually selected keywords (see Appendix C) to infer the operating system and browser family of our browser population. Table 1 compares the share of each operating system and browser family between the previous works that include such statistics [33, 44, 60], the average share of each family between January 2017 and June 2017 measured by StatCounter for the worldwide [90, 91] and French browser populations [3, 4], and our browser population. We stress that the browser population that visits a website varies through time as the browser market evolves [32] and depends on the website (e.g., only a subset of the worldwide browser population is expected to visit a local news website). Among our browser population, 82.1% run on a Windows operating system, resulting in a high proportion of Internet Explorer (26%) and Edge (7.8%) browsers. This can be explained by the population visiting the website being less technically savvy, hence using more common web environments (e.g., a Firefox on a Windows operating system) than technical environments (e.g., Linux-based operating systems).

Among the desktop browsers, the most common browsers are Firefox (31.4%), Internet Explorer (28.3%), and Chrome (26.2%). Our dataset contains more Firefox browsers and less Chrome browsers than the French population observed by StatCounter [3]. According to previous studies, the fingerprints of Firefox and Chrome browsers are among the most distinctive [10, 62]. Firefox and Chrome both include automated updates, which results in higher instability of the fingerprints collected from these browsers [62]. Internet Explorer and Edge supported badly WebRTC at the time of the experiment [5, 10]. On these browsers, no information could have been collected using WebRTC, which reduces the distinctiveness of this attribute on these browsers. However, Edge is considered as one of the browsers that provide highly distinctive fingerprints [10, 62].

The vast majority of our mobile browsers runs on an Android platform (84.4%), followed by Windows Phone (8.8%), and iOS (5.2%). The browsers running on Android devices tend to be more distinguishable than the ones running on iOS [60], due to the plurality of device vendors and models that embark the Android operating system. The browser fingerprints of iOS devices show more updates than these of other mobile operating systems [62], which can result in increased instability of the fingerprints of iOS devices. The most common mobile browser is Samsung Internet (45.1%), followed by Chrome (38.5%), Internet Explorer mobile (8.4%), and Safari (6.7%). Smartphones embark a default browser, which can explain this distribution that includes the four default browsers[1] of the major operating systems. Using another browser than the default browser results in an increased distinctiveness as the combination of operating system and browser becomes less common [62]. Although Chrome is one of the default browser of Android devices, it provides highly distinctive information in its UserAgent [60]. As a result, Firefox and Chrome are known as the two most distinctive mobile browsers [62].

The contextual attributes related to the time zone or to the configured language are less distinctive in our dataset than in previous studies (see Section 5.1.2). For example, the normalized entropy of the `Timezone` JavaScript property is of only 0.008, against 0.161 for the Panopticlick study [33], and 0.198 for the AmIUnique study [60]. These attributes also tend towards the typical French values:

[1]Samsung Internet is the default browser on some Android devices manufactured by Samsung.

98.48% of the browsers have a `Timezone` value of −1, 98.59% of the browsers have the daylight saving time enabled, and `fr` is present in 98.15% of the values of the `Accept-Language` HTTP header.

The browsers of our dataset mostly belong to general audience French users. Counter-intuitively, considering an international population may not reduce the distinctiveness. Indeed, we can expect foreign users to have a combination of contextual attributes (e.g., the time zone, the configured languages) different from the French users, making them distinguishable even if the remaining attributes have identical values. Forging new browser fingerprints from a dataset to realistically simulate a different browser population is a difficult task. First, it requires to know the distribution of the values that the target population would show for each attribute. The value of some attributes can be inferred easily (e.g., the typical time zone of a given location), but the typical value that other attributes would present is harder to infer (e.g., the textual value of the UserAgent of a new browser family). Second, although we could sample the value of each individual attribute from their hypothetical distribution, the value of the attributes should be picked with respect to the correlations that occur between them. When forging a browser fingerprint, the lack of consideration of the correlations between the attributes leads to unrealistic fingerprints [12, 27, 33, 98].

2.3 Privacy concerns

The browser fingerprints are sensitive due to their identification capacity. We complied with the European directives 2002/58/CE [1] and 2009/136/CE [2] in effect at the time of the experiment, and took additional measures to protect the participating users. First, the script was set on two web pages of a single domain in a first-party manner, hence providing no extra information about the browsing activity of the users. The content of the web pages are generic, hence they do not leak any information about the interests of the users. Second, we restricted the collection to the users having consented to cookies, as required by the European directives 2002/58/CE and 2009/136/CE. Third, a unique identifier (UID) was set as a cookie with a 6-months lifetime, corresponding to the duration of the experiment. Fourth, we deleted the fingerprints for which the `cookieEnabled` property was not set to `true`. Finally, we hashed the IP addresses by passing them through the HMAC-SHA256 algorithm using a key that we threw afterward. It was done using the `secret` and the `hmac` libraries of Python3.6. These one-way hashed IP addresses are only used for the UIDs resynchronization (see Section 2.4.2) and are not used as an attribute in the working dataset.

2.4 Data preprocessing

Given the experimental aspect of browser fingerprints, and the scale of our collection, the raw dataset contains erroneous or irrelevant samples. That is why we perform several preprocessing steps before any analysis. The dataset is composed of entries in the form of (f, b, t) tuples so that the fingerprint f was presented by the browser b at the given time t. We talk here about entries (i.e., (f, b, t) tuples) and not fingerprints (i.e., only f) to avoid confusion. The preprocessing is divided in four steps: the cleaning, the UIDs resynchronization, the deduplication, and the derivation of the extracted attributes. Initially, we have 8, 205, 416 entries in the raw dataset.

2.4.1 Dataset cleaning. The dataset cleaning step filters out 70, 460 irrelevant entries, following the method described below. The fingerprinting script prepares, sends, and stores the entries in string format consisting of the attribute values separated by semicolons. We remove 769 entries that have a wrong number of fields, mainly due to truncated or unrelated data (e.g., the body of a post request). We filter out 53, 251 entries that belong to robots, by checking whether blacklisted keywords are present in the `User-Agent` HTTP header (see Appendix C for the list of keywords). We reduce the entries that have multiple exact copies (down to the same moment of collection) to a

single instance. Finally, we remove 18, 591 entries that have the cookies disabled, and 2, 412 entries that have a time of collection that falls outside the time window of the experiment.

2.4.2 Unique IDs resynchronization. The resynchronization step replaces 181, 676 UIDs with a total of 116, 708 other UIDs, following the method described here. Each entry includes a UID which was stored and retrieved using the cookie mechanism of the browser. The cookies are considered an unreliable browser identification solution [62], hence we undergo a cookie resynchronization step similarly to the Panopticlick study [33]. We consider the entries that have the same (fingerprint, IP address hash) pair to belong to the same browser, even if they show different UIDs. We group the entries by their pair of (fingerprint, IP address hash) and assign the entries of each group a single UID taken from those observed for this group. The resynchronization is processed on all the groups at the exception of the groups that show interleaved UIDs for which we do not modify the UID of their entries. These groups contain entries that have the same (fingerprint, IP address hash) pair but which UID values are interleaved (e.g., b_1, b_2, then b_1 again). The interleaved UIDs are a good indicator that several genuine identical browsers operate on the same private network and share the same public IP address [33].

2.4.3 Deduplication. The deduplication step constitutes the biggest cut in our dataset, and filters out 2, 420, 217 entries. To avoid storing duplicates of the same fingerprint observed several times for a browser, the usual method is to ignore a fingerprint if it was already seen for a browser during the collection [33, 60]. Our script collects the fingerprint on each visit, no matter if it was already seen for this browser or not. To stay consistent with common methodologies, we deduplicate the fingerprints offline. For each browser, we hold the first entry that contains a given fingerprint, and ignore the following entries if they also contain this fingerprint. This method takes the interleaved fingerprints into account, which are the fingerprints so that we observe f_1, f_2, then f_1 again. For example, if a browser b has the entries $\{(f_1, b, t_1), (f_2, b, t_2), (f_2, b, t_3), (f_1, b, t_4)\}$, we only hold the entries $\{(f_1, b, t_1), (f_2, b, t_2), (f_1, b, t_4)\}$ after the deduplication step.

We hold the interleaved fingerprints to realistically simulate the state of the fingerprint of each browser through time. We find that 10.59% of our browsers showed at least one case of interleaved fingerprints. The interleaved fingerprints can come from attributes that switch between two values. An example is the screen size that changes when an external screen is plugged or unplugged. Previous studies discarded the fingerprints that were already encountered for a given browser [33, 44, 60], hiding the interleaved fingerprints.

2.4.4 Extracted attributes. We derive 46 extracted attributes of two types from 9 original attributes. First, we have the extracted attributes that are parts of an original attribute, like an original attribute that is composed of 28 triplets of RGB (Red Green Blue) color values that we split in 28 single attributes. Then, we have the extracted attributes that are derived from an original attribute, like the number of plugins derived from the list of plugins. The extracted attributes do not increase the distinctiveness as they come from an original attribute, and they are at most as distinctive as their original attribute. However, the extracted attributes can offer a higher stability than their original attribute, as the latter is impacted by any little change among the extracted attributes. For example, if exactly one of the 28 RGB values changes between two fingerprint observations, the original attribute is counted as having changed, but only one of the extracted attributes will be.

2.4.5 Working dataset. The working dataset obtained after preprocessing the raw dataset contains 5, 714, 738 entries (comprising the identical fingerprints that are interleaved for each browser), with 4, 145, 408 fingerprints (comprising no identical fingerprint for each browser), and 3, 578, 196 distinct fingerprints. They are composed of 216 original attributes and 46 extracted attributes, for a total of 262 attributes. The fingerprints come from 1, 989, 365 browsers, 27.53% of which have multiple

Table 2. Comparison between the datasets of the studies Panopticlick (PTC), AmIUnique (AIU), Hiding in the Crowd (HitC), Long-Term Observation (LTO), Who Touched My Browser Fingerprint (WTMBF), and this study. As the LTO study tests various attribute sets, we present the ranges of the results obtained by their attribute selection method. The symbols - denotes missing information and * denotes deduced information from the paper. The attributes comprise the derived attributes. The fingerprints are counted after data preprocessing.

	PTC [33]	AIU [60]	HitC [44]	LTO [77]	WTMBF [62]	**This study**
Collection period	01-02/2010	11/2014-02/2015	12/2016-06/2017	02/2016-02/2019	12/2017-07/2018	**12/2016-06/2017**
Collection duration	3 weeks	3-4 months*	6 months	3 years	8 months	**6 months**
Attributes	8	17	17	305	35	**262**
Browsers	455,604*	119,818*	-	-	1,329,927	**1,989,365**
Browsers seen multiple times	-	-	-	-	661,827 (49.76%)	**547,672 (27.53%)**
Fingerprints	470,161	118,934	2,067,942	88,088	7,246,618	**4,145,408**
Distinct fingerprints	409,296	142,023[a]	-	9,822–16,541	1,586,719	**3,578,196**
Ratio of desktop fingerprints	-	0.890*	0.879	0.697*–0.707*	-	**0.805**
Ratio of mobile fingerprints	-	0.110*	0.121	0.293–0.303	-	**0.134**
Unicity of overall fingerprints	0.836	0.894	0.336	0.954–0.958	≈ 0.350*	**0.818**
Unicity of mobile fingerprints	-	0.810	0.185	0.916–0.941	≈ 0.350*	**0.399**
Unicity of desktop fingerprints	-	0.900	0.357	0.974–0.978	≈ 0.350*	**0.884**

[a]This number is displayed in Figure 11 of [60] as the number of distinct fingerprints, but it also corresponds to the number of raw fingerprints. Every fingerprint would be unique if the number of distinct and collected fingerprints are equal. Hence, we are not confident in this number, but it is the number provided by the authors.

fingerprints. Table 2 displays a comparison between the dataset of the studies Panopticlick [33], AmIUnique [60], Hiding in the Crowd [44], Long-Term Observation [77], Who Touched My Browser Fingerprint [62], and this study.

2.5 Comparison with previous studies

We compare our working dataset with the datasets of previous studies in Table 2, notably by the unicity rate which is the proportion of the fingerprints that have been observed for a single browser.

We have a slightly lower unicity rate compared to the studies Panopticlick (PTC) [33] and AmIUnique (AIU) [60]. Considering a larger browser population generally increases the chances for two browsers to share the same fingerprint, which can reduce the unicity rate. However, considering a wider surface of fingerprinting attributes usually reduces these chances, which can increase the unicity rate. The importance of these two effects depends on the browser population and on the attributes. The larger browser population and the larger surface of fingerprinting attributes can explain the slight decrease of the unicity rate compared to PTC and AIU. We also observe a lower unicity rate for the fingerprints of mobile browsers compared to the fingerprints of desktop browsers, confirming the findings of previous studies [33, 44, 60, 87]. However, we emphasize that our dataset and the datasets of these studies miss attributes that could improve the distinctiveness of the fingerprints of mobile browsers like the sensor API [103] that is used in real-life [30, 49, 67].

The authors of the Hiding in the Crowd (HitC) study [44] worked on fingerprints collected from the same experiment and browser population as us. However, to stay consistent with their previous AIU study [60], they consider the same set of 17 attributes. This explains our higher number of fingerprints, as two browsers that have different fingerprints for a given set of attributes can come to the same fingerprint if a subset of these attributes is considered. As a result, they remove more duplicated fingerprints than us due to the higher chances for a browser to present the same fingerprint for 17 attributes than for 216 attributes. Our unicity rate is also higher, being at 81.8% for the complete dataset against 33.6% for HitC [44]. This is due to the larger set of considered attributes that distinguish browsers more efficiently, as each additional attribute can provide a way to distinguish browsers. Our little drops on the proportion of desktop and mobile browsers come from a finer-grained classification, as we have 4.8% of the browsers that are classified as belonging to tablets, smart TVs, or game consoles.

The recent study about the evolution of fingerprints by Song Li and Yinzhi Cao [62] analyzes a large-scale dataset of 7, 246, 618 fingerprints collected from a European website. They focus on the evolution of the fingerprints and their composing attributes. Their fingerprints show a low unicity rate of approximately 0.350. The high number of collected fingerprints and the low number of used attributes can explain this lower unicity rate[2]. We remark that this unicity rate is close to the unicity rate of HitC [44] which includes fewer attributes (17 against 35 for [62]) and fewer fingerprints (2, 067, 942). The unicity rate of the mobile browsers of [62] is close to the unicity rate of their desktop browsers, contrary to previous studies [33, 44, 60, 87] and ours. This can be explained by the inclusion of new attributes that provide more distinctiveness to the mobile fingerprints. The attributes that they have in addition to the previous studies[3] are the features that they infer from the IP address. Song Li and Yinzhi Cao [62] focus on the evolution of fingerprints and analyze what they call *dynamics*. They define a dynamic as a representation of the evolutions

[2]The authors of [62] emphasize that the unicity rate is "relatively low because many browsers visited their collection website more than once". This can indicate that they set the identical fingerprints collected from the same browser in an anonymity set. As a result, this set is at least of size 2 (non-unique) whereas the fingerprints may have been seen for this single browser only, which makes them actually unique.

[3]The Panopticlick study [33] leverages IP address hashes to resynchronize fingerprints (see Section 2.4.2) but do not include any information related to the IP address in the fingerprints.

between two different and consecutive fingerprints that belong to the same browser. They generate 960, 853 dynamics from their seven million fingerprints whereas we generate 3, 725, 373 dynamics (see Section 4.2) from our four million fingerprints. This difference is explained by our fingerprints being composed of more attributes, which increases the chances for two consecutive fingerprints to differ. Indeed, the change of any attribute between the consecutive fingerprints generates a dynamic, and the more attributes there is the more chances one of them changes between two evolutions[4].

3 AUTHENTICATION FACTOR PROPERTIES

Biometric authentication factors and browser fingerprints share strong similarities. They both work by extracting features from a unique entity, which is a person for the former and a browser for the latter, that can be used for identification or authentication. Although the entity is unique, the extracted features are a digital representation of the entity which can show imperfections (e.g., the fingerprints of two different persons can show similar representations). Previous studies [39, 66, 107] identified the properties for a biometric characteristic to be *usable*[5] as an authentication factor, and the additional properties for a biometric authentication scheme to be *practical*. We evaluate browser fingerprints according to these properties because of their similarity with biometric authentication factors.

The four properties needed for a biometric characteristic to be *usable* as an authentication factor are the following.

- *Universality*: the characteristic should be present in everyone.
- *Distinctiveness*: two distinct persons should have different characteristics.
- *Permanence*: the same person should have the same characteristic over time. We rather use the term *stability*.
- *Collectibility*: the characteristic should be collectible and measurable.

The three properties that a biometric authentication scheme requires to be *practical* are the following.

- *Performance*: the scheme should be accurate, consume few resources, and be robust against environmental changes.
- *Acceptability*: the users should accept to use the scheme in their daily lives.
- *Circumvention*: it should be difficult for an attacker to deceive the scheme.

The properties that we study are the *distinctiveness*, the *stability*, and the *performance*. We consider that the *universality* and the *collectibility* are satisfied, as the HTTP headers that are automatically sent by browsers constitute a fingerprint. However, we stress that a loss of distinctiveness occurs when no JavaScript attribute is available. About the *circumvention*, we discuss the design of an authentication mechanism that leverages browser fingerprints and the possible attacks on such mechanism in Section 6. As for the *acceptability*, we refer to [17] for a survey about the acceptability of an authentication mechanism that relies on browser fingerprinting, and emphasize that such mechanisms are already used in real-life [37, 85, 105].

[4]Would all the additional attributes never change, their addition would not induce more changes between the consecutive fingerprints. However, our dataset only contains 5 attributes that never presented any change between the three million consecutive fingerprints (see Section 5.1.3).

[5]Here, *usable* refers to the adequacy of the characteristic to be used for authentication, rather than the ease of use by the users.

3.1 Distinctiveness

To satisfy the *distinctiveness* property, the browser fingerprints should distinguish two different browsers. The two extreme cases are every browser sharing the same fingerprint, which makes them indistinguishable from each other, and no two browsers sharing the same fingerprint, making every browser distinguishable. The distinctiveness falls between these extremes, depending on the attributes and the browser population. We consider the use of browser fingerprinting as an additional authentication factor. Hence, we do not require a perfect distinctiveness, as it is used in combination with other authentication factors to improve the overall security.

The dataset entries are composed of a fingerprint, the source browser, and the moment of collection in the form of a Unix timestamp in milliseconds. We denote B the domain of the unique identifiers, F the domain of the fingerprints, and T the domain of the timestamps. The fingerprint dataset is denoted D and is formalized as:

$$D = \{(f, b, t) \mid f \in F, b \in B, t \in T\} \tag{1}$$

We use the size of the browser anonymity sets to quantify the distinctiveness, as the browsers that belong to the same anonymity set are indistinguishable. We denote $\mathcal{B}(f, \mathcal{D})$ the browsers that provided the fingerprint f in the dataset D. It is formalized as:

$$\mathcal{B}(f, \mathcal{D}) = \{b \in B \mid \forall (g, b, t) \in \mathcal{D}, f = g\} \tag{2}$$

We denote $\mathcal{A}(\epsilon, \mathcal{D})$ the fingerprints that have an anonymity set of size ϵ (i.e., that are shared by ϵ browsers) in the dataset D. It is formalized as:

$$\mathcal{A}(\epsilon, \mathcal{D}) = \{f \in F \mid \text{card}(\mathcal{B}(f, \mathcal{D})) = \epsilon\} \tag{3}$$

A common measure of the fingerprint distinctiveness is the *unicity rate* [33, 44, 60], which is the proportion of the fingerprints that were observed for a single browser. We denote $\mathcal{U}(\mathcal{D})$ the unicity rate of the dataset $\mathcal{D}$, which is formalized as:

$$\mathcal{U}(\mathcal{D}) = \frac{\text{card}(\mathcal{A}(1, \mathcal{D}))}{\text{card}(\{(f, b) \mid \exists t \in T, (f, b, t) \in \mathcal{D}\})} \tag{4}$$

Previous studies measured the anonymity set sizes on the whole dataset [33, 44, 60]. We measure the anonymity set sizes on the fingerprints currently in use by each browser, and not on their whole history. Two different browsers, sharing the same fingerprint on different time windows, are then distinguishable (e.g., a browser was updated before the other). Moreover, a browser that runs in a fancy web environment (e.g., having a custom font) and that has several fingerprints in the dataset (e.g., fifty), will bloat the proportion of unique fingerprints and bias the study (e.g., fifty fingerprints are unique whereas they come from a single browser).

We evaluate the anonymity set sizes on the time-partitioned datasets composed of the last fingerprint seen for each browser at a given time. Let $\mathcal{S}_\tau(\mathcal{D})$ be the time-partitioned dataset originating from $\mathcal{D}$ that represents the state of the fingerprint of each browser after τ days. With t_τ the last timestamp of this day, $\mathcal{S}_\tau(\mathcal{D})$ is defined as:

$$\mathcal{S}_\tau(\mathcal{D}) = \{(f_i, b_j, t_k) \in \mathcal{D} \mid \forall (f_p, b_q, t_r) \in \mathcal{D}, b_j = b_q, t_r \leq t_k \leq t_\tau\} \tag{5}$$

3.2 Stability

To satisfy the *stability* property, the fingerprint of a browser should stay sufficiently similar between two observations to be recognizable. Browser fingerprints have the particularity of evolving through time, due to changes in the web environment like a software update or a user configuration. We measure the stability by the average similarity between the consecutive fingerprints of browsers, given the elapsed time between their observation. The two extreme cases are every browser

holding the same fingerprint through its life, and the fingerprint changing completely with each observation. A lack of stability makes it harder to recognize the fingerprint of a browser between two observations.

We denote $C(\Delta, \mathcal{D})$ the function that provides the pairs of consecutive fingerprints of $\mathcal{D}$ that are separated by a time-lapse comprised in the Δ time range. It is formalized as:

$$C(\Delta, \mathcal{D}) = \{(f_i, f_p) \mid \forall((f_i, b_j, t_k), (f_p, b_q, t_r)) \in \mathcal{D}^2, b_j = b_q, t_k < t_r, (t_r - t_k) \in \Delta, \nexists(f_c, b_d, t_e) \in \mathcal{D}, b_d = b_j, f_c \neq f_i, f_c \neq f_p, t_k < t_e < t_r\} \tag{6}$$

We consider the Kronecker delta $\delta(x, y)$, being 1 if x equals y and 0 otherwise. We consider the set Ω of the n used attributes. We denote $f[\omega]$ the value taken by the attribute ω for the fingerprint f. Let $\text{sim}(f, g)$ be the proportion of identical attributes between the fingerprints f and g, which is formalized as:

$$\text{sim}(f, g) = \frac{1}{n} \sum_{\omega \in \Omega} \delta(f[\omega], g[\omega]) \tag{7}$$

We define the function $\text{avsim}(\Delta, \mathcal{D})$ that provides the average similarity between the pairs of the consecutive fingerprints, for a given time range Δ and a dataset D. It is formalized as:

$$\text{avsim}(\Delta, \mathcal{D}) = \frac{\sum_{(f,g) \in C(\Delta, \mathcal{D})} \text{sim}(f, g)}{\text{card}(C(\Delta, \mathcal{D}))} \tag{8}$$

3.3 Performance

We consider four aspects of the *performance* of browser fingerprints for web authentication: their *collection time*, their *size* in memory, the *loss of efficacy* between different device types, and the *accuracy* of a simple illustrative verification mechanism. Browser fingerprinting can easily be deployed by adding a script on the authentication page, and by preparing servers to handle the reception, the storage, and the verification of the fingerprints. The users solely rely on their regular web browser and do not have to run any dedicated application, nor possess specific hardware, nor undergo a configuration step. The main additional load is on the supplementary consumption of memory and time resources. Moreover, the web environment differs between device types (e.g., mobile browsers have more limited functionalities than desktop browsers) and through time (e.g., modern browsers differ from the browsers from ten years ago), impacting the efficacy of browser fingerprinting. Finally, we evaluate the accuracy of a simple illustrative verification mechanism under fingerprints evolution.

3.3.1 Collection time. The browser fingerprints can be solely composed of passive attributes (e.g., HTTP headers) that are transmitted along with the communications with the server. In this case, the fingerprints are collected without the user perceiving any collection time, but major attributes are set aside. The client-side properties collected through JavaScript provide more distinctive attributes, at the cost of an additional collection time. We measure the collection time of the fingerprints considering only the JavaScript attributes, and ignore the HTTP headers that are transmitted passively.

3.3.2 Size. Browser fingerprinting consumes memory resources on the clients during the buffering of the fingerprints, on the wires during their sending, and on the servers during their storage. The memory consumption depends on the storage format of the fingerprints. For example, a canvas can be stored as an image encoded in a base64 string, or as a hash which is shorter. A trade-off has to be done between the quantity of information and the memory consumption. The less memory-consuming choice is to store the complete fingerprint as a single hash. However, the fingerprints evolve through time – even more when they are composed of many attributes – which results in

the use of a hash of the complete fingerprint being impractical. Due to the unspecified size of the attributes (e.g., the specification of the `User-Agent` HTTP header does not define a size limit [81]), we measure their *size* given a fingerprint dataset.

3.3.3 Loss of efficacy. The *loss of efficacy* is the loss of stability, distinctiveness, or performance of the fingerprints. It can occur either for a group of browsers (e.g., mobile browsers) or resulting from changes brought to web technologies. First, previous works showed differences in the properties of the fingerprints coming from mobile and desktop devices [44, 60, 87], notably a limited distinctiveness for the mobile browsers. Following these findings, we compare the properties shown by the mobile and the desktop browsers. We match keywords on the `userAgent` JavaScript property (see Appendix C) to differentiate the browsers running on a desktop or a laptop (referred to as *desktops*) from the browsers running on a mobile phone (referred to as *mobiles*). Second, browser fingerprinting is closely dependent on the evolution of web technologies. As new technologies are integrated into browsers, new attributes are accessible and conversely for removal. Similarly, functionality updates can lead to a change in the fingerprint properties. For example, Kurtz et al. [55] detected an iOS update by the sudden instability of an attribute that provides the iOS version. Following their finding, we verify whether the evolution of web technologies provokes major losses in the properties of the fingerprints.

3.3.4 Accuracy of a simple verification mechanism. We evaluate the accuracy of a simple illustrative verification mechanism under the evolution of fingerprints. This mechanism counts the identical attributes between the presented and the stored fingerprint, and considers the evolution legitimate if this number is above a threshold Θ. The simplicity of this mechanism gives us an idea of the accuracy that can be easily achieved without having to engineer more complex rules. More elaborate mechanisms can obviously be designed (see Appendix D).

4 EVALUATION OF BROWSER FINGERPRINTS PROPERTIES

In this section, we evaluate the browser fingerprints of our dataset according to the properties of distinctiveness, stability, and performance. We present here the results on the complete fingerprints and let Section 5 provide insights on the contribution of the attributes to each property. We show that, by time-partitioning our dataset, our fingerprints provide a unicity rate of more than 81.3% which is stable through time. However, our fingerprints of mobile browsers are less distinctive than our fingerprints of desktop browsers[6], with a respective unicity rate of 42% against 84% considering the time-partitioned datasets. We also show that, on average, a fingerprint has more than 91% of identical attributes between two observations, even when they are separated by nearly 6 months. About their performance, we show that our fingerprints weigh a dozen of kilobytes and are collected within seconds. The accuracy of the simple illustrative verification mechanism is close to perfect, as it achieves an equal error rate of 0.61%. This results from most of the consecutive fingerprints coming from a browser having at least 234 identical attributes, whereas most of the fingerprints of different browsers have fewer.

4.1 Distinctiveness

4.1.1 Overall distinctiveness. Figure 2 presents the size of the anonymity sets alongside the frequency of browser arrival for the time-partitioned datasets. The time-partitioned datasets are designed so that each browser has the last fingerprint observed at the end of the τ-th day. The

[6]Our study is in line with previous studies [33, 44, 60, 87] about the fingerprints of mobile browsers showing less distinctiveness. However, we emphasize that our dataset and the datasets of these studies miss attributes that could improve the distinctiveness of the fingerprints of mobile browsers like the sensor API [103] that is used in real-life [30, 49, 67].

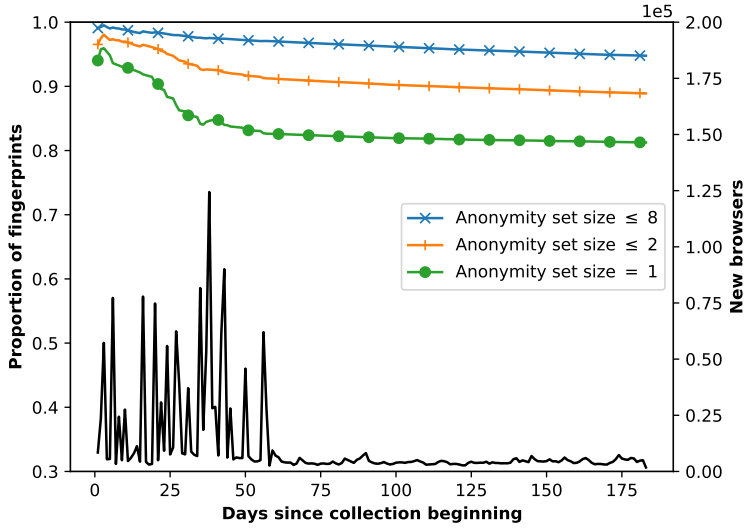

Fig. 2. Anonymity set sizes and frequency of browser arrivals through the time-partitioned datasets obtained after each day. The new browsers are displayed in hundreds of thousands.

overall fingerprints have a stable unicity rate of more than 81.3% for the partitioned-datasets, and more than 94.7% of the fingerprints are shared by at most 8 browsers. The overall fingerprints comprise those collected from desktop and mobile browsers, but also those of tablets, game consoles, and smart TVs. The comparisons are done using fingerprint hashes, resulting in 4 hash collisions which we deem negligible.

The anonymity sets tend to grow as more browsers are encountered due to the higher chances of collision. However, the fingerprints tend to stay in small anonymity sets, as can be seen by the growth of the anonymity sets of size 2 being more important than the growth of the anonymity sets of size 8 or higher. The unicity rate of the time-partitioned datasets (81.3%) is lower than the unicity rate of the complete dataset (81.8%). This is due to browsers having multiple unique fingerprints in the complete dataset, which typically occurs when a browser having a unique web environment is fingerprinted multiple times. Considering the time-partitioned datasets removes this over-counting effect.

New browsers are encountered continually. However, starting from the 60th day, the arrival frequency stabilizes around 5,000 new browsers per day. Before this stabilization, the arrival frequency is variable and has major spikes that seem to correspond to events that happened in France. These events could lead to more visits and explain these spikes. For example, the spike on the 38th day corresponds to a live political debate on TV, and the spike on the 43rd correlates with the announcement of a cold snap.

4.1.2 Distinctiveness of desktop and mobile browsers. Figure 3 presents the unicity rate through the time-partitioned datasets for the overall, the mobile, and the desktop browsers. The fingerprints of mobile browsers are more uniform than those of desktop browsers, with a unicity rate of approximately 42% against 84% considering the time-partitioned datasets. The unicity rate of the desktop browsers slightly increases by 1.04 points from the 60th to the 183th day, from 84.99% to 86.03%. On the contrary, the unicity rate of the mobile browsers slightly decreases by 0.29 points on the same period, from 42.42% to 42.13%. These results are in line with previous studies [33, 44, 60, 87] that also observed a reduction of the distinctiveness among the fingerprints of mobile browsers

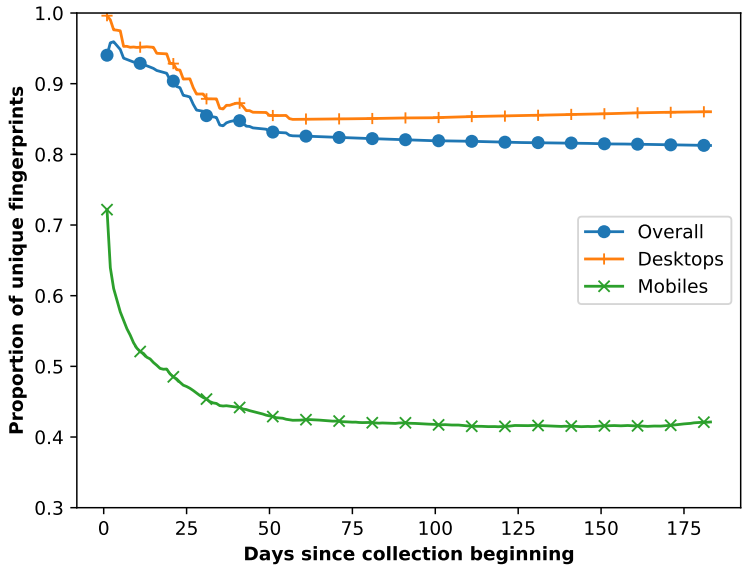

Fig. 3. Unicity rate for the overall, the mobile, and the desktop browsers, through the time-partitioned datasets obtained after each day.

compared to those of desktop browsers. However, we acknowledge that our dataset misses attributes that could improve the distinctiveness of the fingerprints of mobile browsers. A good candidate attribute is the sensor API [103] that is already used by real-life fingerprinters [30, 49, 67].

4.2 Stability

Figure 4 displays the average similarity between the pairs of consecutive fingerprints as a function of the time difference, together with the number of compared pairs for each time difference. The ranges Δ are expressed in days, so that the day d on the x-axis represents the fingerprints that are separated by $\Delta = [d; d+1[$ days. We ignore the comparisons of the time ranges that have less than 10 pairs to have samples of sufficient size without putting too many comparisons aside. We also ignore the bogus comparisons that have a time difference higher than the limit of our experiment (182 days). These two sets of ignored comparisons account for less than 0.03% of each group. The results are obtained by comparing a total of 3, 725, 373 pairs of consecutive fingerprints, which include 2, 912, 860 pairs for the desktop browsers and 594, 591 pairs for the mobile browsers. Two consecutive fingerprints are necessarily different as we remove the duplicated consecutive fingerprints (see Section 2.4.3). Considering (f_1, f_2, f_3) the fingerprints collected for a browser that are ordered by the time of collection. The resulting set of consecutive fingerprints is $\{(f_1, f_2), (f_2, f_3)\}$. Our stability results are a lower bound, as the consecutive fingerprints are necessarily different (i.e., their similarity is strictly lower than 1).

Our fingerprints are stable, as on average more than 91% of the attributes are expected to not change, considering up to 174 elapsed days (nearly 6 months) between two observations. In a web authentication context, we assume that the users would connect more frequently or would accept to undergo the account recovery process (see Section 6.1). The fingerprints of mobile browsers are generally more stable than those of desktop browsers, as suggests their respective similarity curve. Few attributes of our script are highly instable. They are discussed in Section 5.1.3. Getting rid

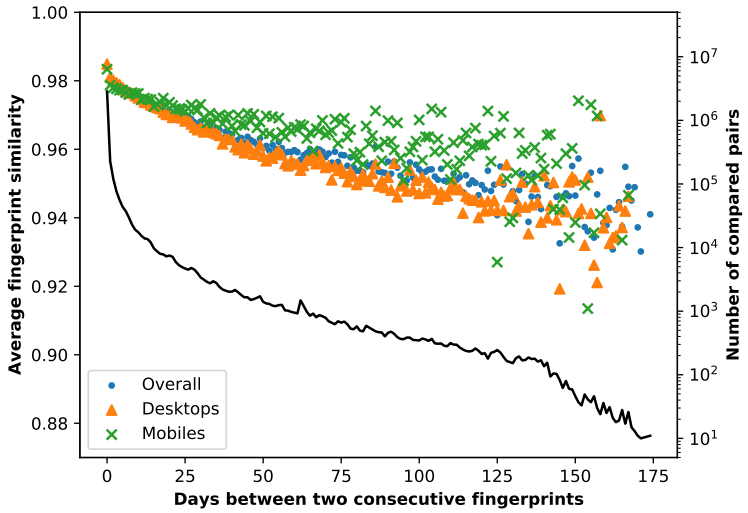

Fig. 4. Average similarity between the pairs of consecutive fingerprints as a function of the time difference, with the number of compared pairs, for the overall, the mobile, and the desktop browsers.

of these attributes could reduce the distinctiveness of the fingerprints, but would improve their stability.

4.3 Performance

4.3.1 Time consumption. Figure 5 displays the cumulative distribution of the collection time of our fingerprints in seconds with the outliers removed. We measure the collection time by the difference between two timestamps, one recorded at the starting of the script and the other one just before sending the fingerprint. Some fingerprints take a long time to collect that span from several hours to days. We limit the results to the fingerprints that take 30 seconds or less to collect and consider the higher values as outliers. The outliers account for less than 1% of each group and are discussed in Appendix B.

Half of our fingerprints are collected in less than 2.92 seconds, and the majority (95%) in less than 10.42 seconds. The time to collect the fingerprints is lower for the desktop browsers than for the mobile browsers. Half of the fingerprints of the desktop browsers are collected in less than 2.64 seconds, and the majority (95%) in less than 10.45 seconds. These numbers are respectively of 4.44 seconds and 10.16 seconds for the mobile browsers. The median collection time of our fingerprints is less than the estimated median time taken by web pages to load completely [19], which is 6.5 seconds for the desktop browsers and 17.9 seconds for the mobile browsers at the date of May 1, 2021. Mobile devices generally have less computing power than desktop devices, which can explain the longer collection time together with the throttling of inactive tabs. The collection time of the fingerprints of mobile browsers has less variance than those of desktop browsers. This can be explained by the former having more uniform computing power than the latter. This is supported by the presence in our dataset of desktop browsers running on old systems like Windows Vista or Windows XP.

Our script takes several seconds to collect the attributes that compose the fingerprints. However, we stress that this script is purely experimental and was developed to collect many attributes to get closer to what a fingerprinter may achieve in real-life. Although collecting the fingerprint induces an additional collection time on the authentication page, such page is expected to be lightweight

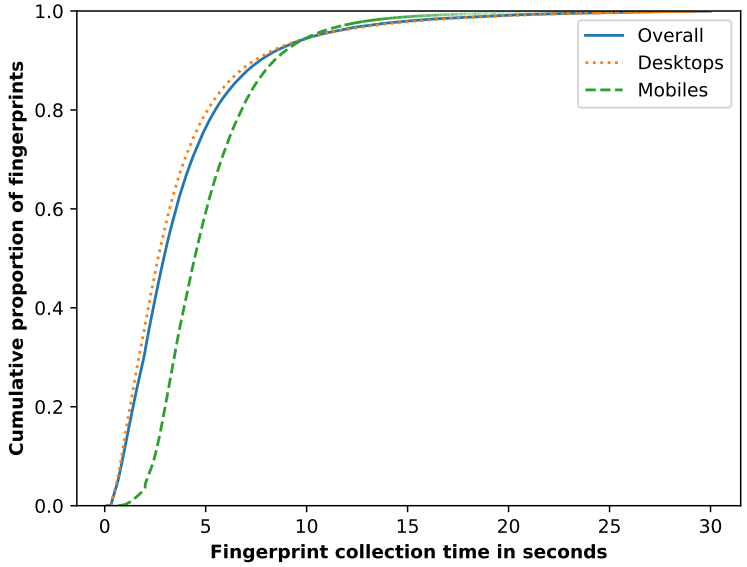

Fig. 5. Cumulative distribution of the collection time of the fingerprints in seconds. We consider the fingerprints that take more than 30 seconds to collect as outliers, which account for less than 1% of each group.

and fast to load. Moreover, the time to collect the fingerprint can be reduced by ways that we describe below. The attributes that are longer to collect and that are less distinctive can be removed without a major loss of distinctiveness [15]. For example, our method to detect an ad-blocker waits a few seconds for a simulated advertisement to be removed, but only provides a Boolean value. We discuss the longest attributes to collect in Section 5.1.4. The attributes can also be collected in parallel or in the background. In their recent work, Song Li and Yinzhi Cao [62] collect their attributes in parallel and manage to collect the fingerprints of desktop and mobile browsers within one second. Our script can also be updated to leverage the most advanced web technologies, like the OffscreenCanvas API [40] that migrates the generation of the canvases off the main thread to another thread. More generally, we can use the Service Workers API [72] to collect the attributes concurrently in the background to reduce the perceived collection time.

4.3.2 Memory consumption. Figure 6 displays the cumulative distribution of the size of our fingerprints in bytes with the outliers removed. Our fingerprints are encoded in UTF-8 using only ASCII characters and the canvases are stored as sha256 hashes. One character then takes one byte and the results can be expressed in both units. The fingerprint sizes comprise the value of the 262 attributes without the metadata fields (e.g., the UID, the timestamp). The average fingerprint size is of $\mu = 7,692$ bytes, and the standard deviation is of $\sigma = 2,294$. We remove 1 fingerprint from a desktop browser considered an outlier due to its size being greater than $\mu + 15 \cdot \sigma$.

The memory consumption takes place on three components: on the client during the buffering of the fingerprints, on the wire during their sending, and on the server during their storage. Half of our fingerprints take less than 7,550 bytes, 95% less than 12 kilobytes, and all of them less than 22 kilobytes. This is negligible given the current storage and bandwidth capacities. We observe a difference between the fingerprints of mobile and desktop browsers, with 95% of the fingerprints weighing respectively less than 8,020 bytes and 12,082 bytes. This is due to heavy attributes being

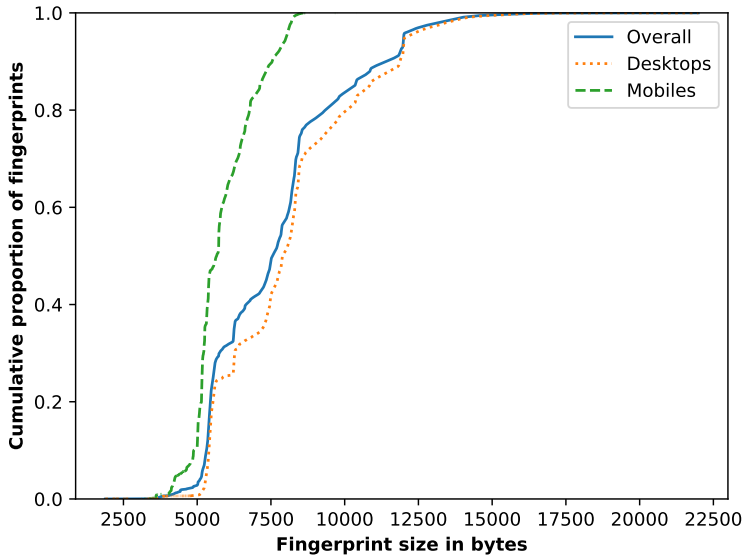

Fig. 6. Cumulative distribution of the fingerprint size in bytes, for the overall, the mobile, and the desktop browsers. We ignore one outlier collected from a desktop browser that shows an extremely higher size.

lighter on mobile browsers, like the list of plugins or mime types that are most of the time empty. We discuss the heaviest attributes in Section 5.1.5.

4.3.3 Accuracy of the simple verification mechanism. The accuracy of the simple illustrative verification mechanism is measured according to the following methodology. First, we split our dataset in *six samples*, one for each month. We assume that a user would spend at most one month between two connections, and otherwise would accept to undergo a heavier fingerprint update process. Two sets are afterward extracted from each sample. They are composed of pairs of compared fingerprints that we call *comparisons*. The *same-browser* comparisons are composed of the consecutive fingerprints of each browser, and the *different-browsers* comparisons are composed of two randomly picked fingerprints of different browsers. After constituting the same-browser comparisons for each month, we sample the different-browsers comparisons to have the same size as the same-browser comparisons. The month sampling also helps the different-browsers comparisons to be realistic by pairing fingerprints that are separated by at most one month. Both the two sets of comparisons contain a total of 3, 467, 289 comparisons.

Figure 7 displays the distribution of the identical attributes between the same-browser comparisons and the different-browsers comparisons, starting from 34 identical attributes as there are no observed value below. Figure 8 presents a focus that starts from 227 identical attributes, below which there are less than 0.005 of the same-browser comparisons. We can observe that the two sets of comparisons are well separated, as 99.05% of the same-browser comparisons have at least 234 identical attributes, and 99.68% of the different-browsers comparisons have fewer. The different-browsers comparisons have generally a fewer, and a more diverse, number of identical attributes compared to the same-browser comparisons. The different-browsers comparisons have between 34 and 253 identical attributes, with an average of 127.41 attributes and a standard deviation of 44.06 attributes. The same-browser comparisons have between 72 and 252 identical attributes, with an average of 248.64 attributes and a standard deviation of 3.91 attributes.

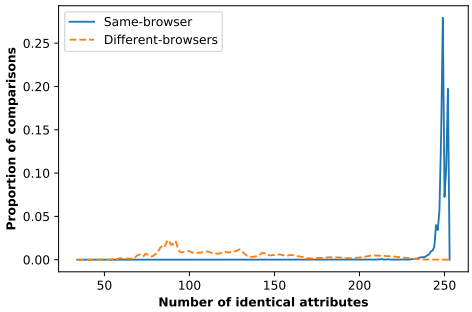

Fig. 7. The number of identical attributes between the same-browser comparisons and the different-browsers comparisons.

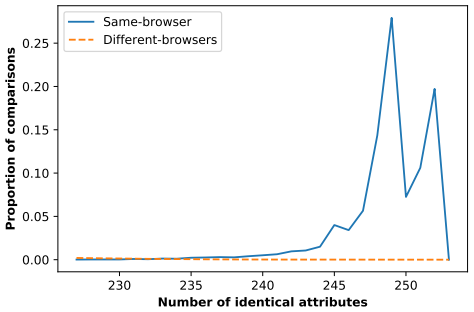

Fig. 8. The number of identical attributes between the same-browser comparisons and the different-browsers comparisons, starting from 227 attributes.

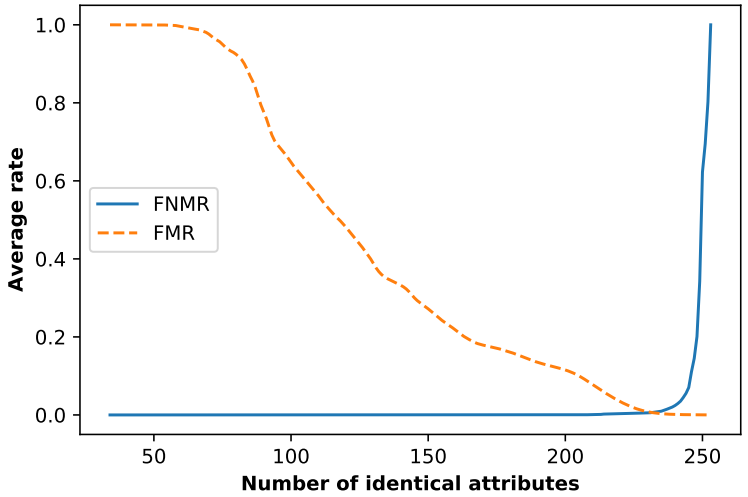

Fig. 9. False match rate (FMR) and false non-match rate (FNMR) given the required number of identical attributes, averaged among the month six samples.

Figure 9 displays the false match rate (FMR) which is the proportion of the same-browser comparisons that are classified as different-browsers comparisons, and the false non-match rate (FNMR) which is the inverse. The displayed results are the average for each number of identical attributes among the six month-samples. As there are few same-browser comparisons that have less than 234 identical attributes, the FNMR is null until this value. However, after exceeding this threshold, the FNMR increases as the same-browser comparisons begin to be classified as different-browsers comparisons. The equal error rate, which is the rate where both the FMR and the FNMR are equal, is of 0.61% and is achieved for 232 identical attributes. Although the verification mechanism does not have a perfect accuracy of 100%, this is acceptable. Indeed, a user getting his browser unrecognized can undergo the fallback authentication process (see Section 6.1). Moreover, we consider the use of browser fingerprinting as an additional authentication factor, hence the other factors can prevent a falsely recognized browser. Both these events are expected to rarely occur as can be seen by the low equal error rate.

These results are tied to the distinctiveness and the stability of the fingerprints. Indeed, as more than 94.7% of the fingerprints are shared by less than 8 browsers, two random fingerprints have little chances to match. Moreover, more than 244 attributes (96.64%) are identical between the consecutive fingerprints of a browser[7], on average and when separated by less than 31 days. This is consistent with the 248.64 identical attributes on average among the same-browser comparisons[8].

4.4 Conclusion

About the distinctiveness, and considering the time-partitioned datasets, our fingerprints provide a unicity rate of more than 81.3% which is stable on the long-run. Moreover, more than 94.7% of our fingerprints are shared by at most 8 browsers. About the stability, and on average, a fingerprint has more than 91% of its attributes that stay identical between two observations, even when they are separated by nearly 6 months. About the performance, the majority (95%) of our fingerprints weigh less than 12 kilobytes and are collected in less than 10.42 seconds. We do not remark any significant loss in the properties offered by our fingerprints through the 6 months of our experiment. However, we fall to the same conclusion as previous studies [33, 44, 60, 87] about the fingerprints of mobile browsers lacking distinctiveness. Their unicity rate in the time-partitioned datasets falls down to approximately 42%. We remark that, in our dataset, the consecutive fingerprints of a browser have at least 234 identical attributes, whereas the majority of the fingerprints of different browsers have fewer. This results in our simple verification mechanism achieving an equal error rate of 0.61%.

5 ATTRIBUTE-WISE ANALYSIS

In this section, we discuss the contribution of the attributes to the properties of the fingerprints. Then, we show that most of the attributes are correlated with at least another one (i.e., they provide less than 1 bit of entropy when the other one is known). Finally, we focus on the properties of the dynamic attributes (e.g., their collection time). We refer the reader to Appendix A for more information about the implementation of each attribute.

5.1 Contribution of particular attributes

In this section, we discuss the contribution of the attributes to the fingerprint properties of distinctiveness, stability, and performance. We express the stability of the attributes as the *sameness rate*, which is the proportion of the consecutive fingerprints where the value of the attribute stays identical. Appendix E provides the exhaustive list of the attributes and their properties.

5.1.1 Attributes distinct values. Figure 10 depicts the cumulative distribution of the number of distinct values among the attributes. We have 42% of the attributes that have at most 10 distinct values, 63% that have at most 100 distinct values, and 79% that have at most 1,000 distinct values. Only 5 attributes have more than 100,000 distinct values. They are the WebRTC fingerprinting method (671,254 values), the list of plugins (314,518 values), the custom canvas in the PNG format (269,874 values) and in the JPEG format (205,005 values), together with the list of mime types (174,876 values).

The values of the attributes are of different nature, impacting the distinct values that can be observed among different population or time ranges. Some attributes have a *fixed number of values*, like the Boolean attributes or the categorical attributes that have a fixed set of possibilities. Other

[7]On average, more than 90% of the attributes stay identical between the consecutive fingerprints, but these attributes are not always the same. We do not restrict the pairs of consecutive fingerprints to the subset of the identical attributes as these attributes differ between these pairs.

[8]The difference is due to the fingerprints of the same-browser comparisons belonging to the same month, whereas the consecutive fingerprints can overlap over two consecutive months.

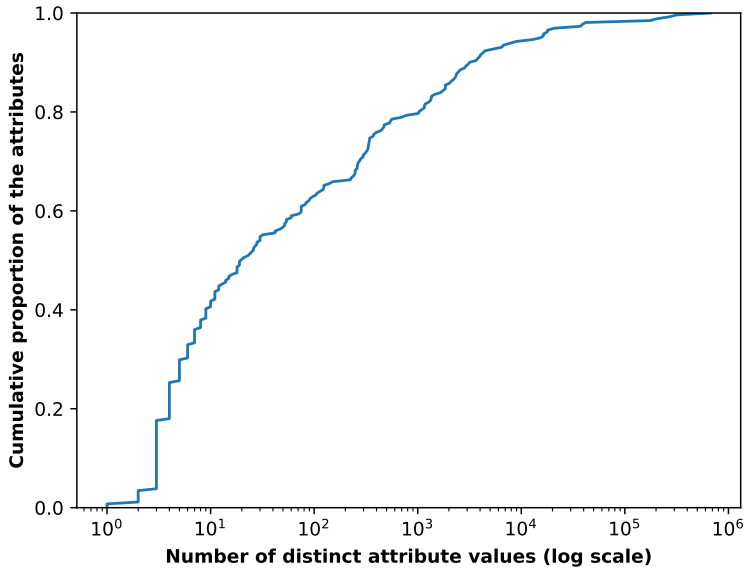

Fig. 10. Cumulative distribution of the number of distinct values of the attributes in logarithmic scale.

attributes are composed of *elements having a fixed set* of possibilities, but their combination provides a high number of values. It is the case for the attributes that are related to languages that typically are a combination of language identifiers (e.g., fr) that can be weighted (e.g., $q = 0.80$). These attributes also comprise the list of fonts that is composed of Boolean values that indicates the presence of a font. Other attributes are *integers* or *floating-point numbers*, resulting in a large set of possible values. This category comprises any size-related attribute (e.g., the screen width and height) and our audio fingerprinting method which value is a floating-point number with a high precision of more than 8 digits. Other attributes are *textual information* that can include version numbers, which results in a high number of distinct values. Moreover, as new values appear through time (e.g., new versions, new software components), the set of the observed values is expected to grow over the observations. Examples are the UserAgent or the list of plugins which are composed of the name and version of hardware and software components.

Table 3 compares the number of distinct attributes of our dataset with the numbers reported by the studies AmIUnique [60], Hiding in the Crowd [44], and Who Touched My Browser Fingerprint [62]. We emphasize that the more fingerprints are observed, the more chances there is to observe new attribute values. We find two attributes missing from previous studies that show a high number of distinct values: the WebRTC fingerprinting method which includes numerous information about an instance of a current WebRTC connection, and the list of mime types.

Three attributes have a higher number of distinct values in our study than in the comparison studies [44, 60, 62]: the list of plugins, the canvas, and the list of HTTP headers. For the list of plugins, this increase can be explained by the higher number of observed fingerprints compared to [60] and by the higher proportion of desktop browsers[9] compared to [62]. The increase of distinct values of the canvas is due to the larger set of instructions that we use for the custom canvas compared to the comparison studies (see Section 5.3). The increase of distinct values of the HTTP

[9]Due to a lack of personalization, the browsers that run on mobile devices tend to have fewer plugins than those that run on desktops or laptops [44, 60].

Table 3. Comparison of the distinct attribute values between the studies AmIUnique (AIU), Hiding in the Crowd (HitC), Who Touched My Browser Fingerprint (WTMBF), and this study. The attributes are composed of the five attributes showing the most distinct values for this study and the attributes reported by the previous studies that are in this study. The attributes are ranked according to this study. The symbol - denotes missing information and * denotes the attributes that are not collected exactly the same way (e.g., using different set of instructions or methods) among the studies.

Attribute	AIU [60]	HitC [44]	WTMBF [62]	**This study**
WebRTC	-	-	-	**671,254**
List of plugins*	47,057	288,740	16,633	**314,518**
Canvas (PNG)*	8,375	78,037	14,006	**269,874**
Canvas (JPEG)	-	-	-	**205,005**
List of mime types	-	-	-	**174,876**
User agent	11,237	19,775	41,060	**20,961**
List of fonts*	36,202	17,372	115,128	**17,960**
List of HTTP headers*	1,182	610	344	**5,394**
WebGL renderer (unmasked)	1,732	3,691	5,747	**3,786**
Content language	4,694	2,739	14,214	**2,833**
Pixel ratio	-	-	1,930	**2,035**
WebGL canvas*	-	-	4,152	**1,158**
CPU cores	-	-	29	**60**
Timezone	55	60	38	**60**
Platform	187	32	-	**32**
Content encoding	42	30	26	**30**
WebGL vendor (unmasked)	26	27	26	**27**
Accept	131	24	9	**26**
Do Not Track	7	3	-	**11**
AdBlock*	2	2	-	**19**
Color depth	-	-	6	**14**
CPU class	-	-	5	**6**
Use of local storage	2	2	2	**4**
Use of session storage	2	2	-	**4**
Cookies enabled	2	1	2	**1**
Fingerprints	118,934	2,067,942	7,246,618	**4,145,408**
Distinct fingerprints	142,023[a]	-	1,586,719	**3,578,196**

[a]This number is displayed in Figure 11 of [60] as the number of distinct fingerprints, but it also corresponds to the number of raw fingerprints. Every fingerprint would be unique if the number of distinct and collected fingerprints are equal. Hence, we are not confident in this number, but it is the number provided by the authors.

headers list is explained by the comparison studies storing the name of the headers, whereas we store both the name and the value of the headers that we do not store in a dedicated attribute. Other attributes have a higher number of distinct attributes because we store a flag that indicates the type of error when one is encountered (e.g., the value was not collected in time, its collection raised an exception). For example, the support of the local storage has 4 distinct values instead of 2 for the previous studies [44, 60]. Moreover, for the detection of an ad-blocker, we verify several

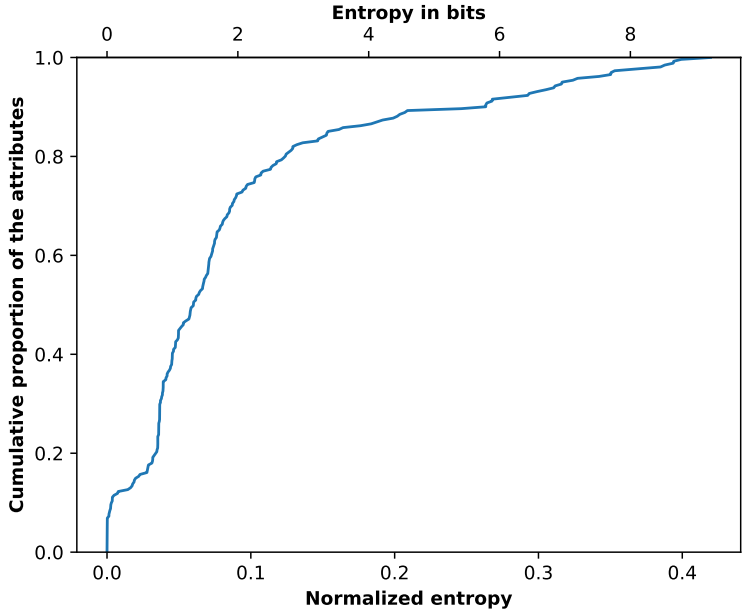

Fig. 11. Cumulative distribution of the normalized entropy and entropy (in bits) among the attributes.

modifications that a potential ad-blocker would make on a dummy advertisement (e.g., set its visibility to hidden) and store a list of whether each modification was done.

Four attributes have more distinct values in the study of Laperdrix et al. [60]: the list of fonts, the platform, and three HTTP headers that are accept, content language, and content encoding. The increase for the list of fonts is due to their usage of the Flash plugin that directly accesses the list of installed fonts. Due to the deprecation of plugins, we infer the presence of 66 fonts through the size of text boxes [35]. The increase for the four other attributes can be explained by the diversity of the configurations of their browser population and by users modifying their browser fingerprint (e.g., they found users modifying the platform attribute and counted 5, 426 inconsistent fingerprints).

Four attributes have more distinct values in the study of Song Li and Yinzhi Cao [62]: the user agent, the list of fonts, the content language, and the WebGL canvas. The increase of distinct values of the user agent can be explained by their higher proportion of mobile browsers, which tend to have more information in the user agent (e.g., the device model [60]). The increase for the list of fonts can be explained by their higher number of fonts that are detected, which is of 96 fonts compared to 66 in our study. As for the content language, they collect their fingerprints from a European website and observe connections coming from 226 countries. Although they remark that some connections are made through proxies, we can still suppose that the user population span over several countries. As a result, their dataset contains more diverse language configurations than our dataset which was collected from a French website. The increase of distinct values of the WebGL canvas can be explained by their set of instructions being more complex than ours (see Section 5.3). Unfortunately, they do not provide any example of generated images or description of the instructions they used.

5.1.2 Attributes distinctiveness. We measure the distinctiveness of the attributes as the *normalized entropy* for comparability with previous studies. The normalized entropy was proposed by Laperdrix et al. [60] to cope with the problem of comparing the entropy of attributes between

Table 4. Comparison of the normalized entropy between the studies Panopticlick (PTC), AmIUnique (AIU), Hiding in the Crowd (HitC), and this study. The attributes are ranked from the most to the least distinctive in this study. The symbol - denotes missing information and * denotes the attributes that are not collected exactly the same way (e.g., using different set of instructions or methods) among the studies.

Attribute	PTC [33]	AIU [60]	HitC [44]	**This study**
Canvas (PNG)*	-	0.491	0.407	**0.420**
Canvas (JPEG)	-	-	0.391	**0.399**
List of plugins*	0.817	0.656	0.452	**0.394**
User agent	0.531	0.580	0.341	**0.350**
List of fonts*	0.738	0.497	0.329	**0.305**
WebGL renderer (unmasked)	-	0.202	0.264	**0.268**
Content language	-	0.351	0.129	**0.124**
WebGL vendor (unmasked)	-	0.127	0.109	**0.115**
List of HTTP headers*	-	0.249	0.085	**0.095**
Do Not Track	-	0.056	0.091	**0.085**
Platform	-	0.137	0.057	**0.068**
Accept	-	0.082	0.035	**0.028**
Content encoding	-	0.091	0.018	**0.019**
Timezone	0.161	0.198	0.008	**0.008**
AdBlock*	-	0.059	0.002	**0.002**
Use of local storage	-	0.024	0.002	**0.001**
Use of session storage	-	0.024	0.002	**0.000**
Cookies enabled	0.019	0.015	0.000	**0.000**
Maximum entropy H_M	18.843	16.859	20.980	**21.983**
Fingerprints	470,161	118,934	2,067,942	**4,145,408**

fingerprint datasets of dissimilar sizes. The normalized entropy $H_n(X)$ is defined as the ratio of the entropy $H(X)$ of the attribute to the maximum entropy $H_M = \log_2(N)$, with N being the number of fingerprints. The entropy can be retrieved as $H(X) = H_n(X) * H_M$. In our case, $H_M = 21.983$ bits and an entropy of 1 bit is equivalent to a normalized entropy of 0.045.

Figure 11 displays the cumulative distribution of the normalized entropy and entropy (in bits) among the attributes. We have 10% of the attributes that provide a low normalized entropy of less than 0.003, and another 10% that provide a normalized entropy between 0.25 and 0.42. The majority of the attributes (80%) provide a normalized entropy comprised between 0.003 and 0.25. The most distinctive attributes of previous studies [33, 60] also belong to the most distinctive attributes of our study. The three *canvases* are among the 10 most distinctive attributes. Our designed canvas in PNG has a normalized entropy of 0.420, the canvas similar to the canvas presented in the Morellian study [56] has a normalized entropy of 0.385, and the canvas inspired by the AmIUnique study [60] has a normalized entropy of 0.353. The *userAgent* collected from the JavaScript property is more distinctive than its HTTP header counterpart, as they respectively have a normalized entropy of 0.394 and 0.350. Finally, the *list attributes* are also highly distinctive. The list of plugins has a normalized entropy of 0.394, the list of supported mime types has a normalized entropy of 0.311, and the list of fonts has a normalized entropy of 0.305.

Table 4 compares the normalized entropy between the studies Panopticlick [33], AmIUnique [60], Hiding in the Crowd [44], and this study. The attributes are ranked from the most to the least

distinctive in this study. The normalized entropy of our attributes are lower than what is reported in these studies, which can be explained by the following factors. First, the maximum entropy H_M increases with the number of fingerprints. However, an attribute that has n possibilities (e.g., a Boolean attribute has only two possible values) have a normalized entropy of at most $\log_2(n)$. As the number N of fingerprints increases, the normalized entropy decreases due to the entropy of the attribute being capped at $\log_2(n)$ whereas the ratio is to $\log_2(N)$. Second, contextual attributes are biased towards the French population (see Section 2.2) and provide a lower normalized entropy. For example, the time zone and the `Accept-Language` HTTP header (named content language in Table 4) provide a respective normalized entropy of 0.008 and 0.124, against 0.198 and 0.351 for the AmIUnique study. The third reason is the evolution of the web technologies since the Panopticlick and the AmIUnique study. For example, the list of plugins is less distinctive due to the replacement of plugins by HTML5 functionalities or extensions [94]. Another example is the list of fonts that was collected through plugins [33, 60], but now has to be inferred from the size of text boxes [35]. Although the 17 attributes in common have a lower normalized entropy, we supplement them with more than a hundred attributes, resulting in the fingerprints showing a unicity rate above 80%.

Interestingly, four attributes unreported by the previous large-scale studies [33, 44, 60] are found to be highly distinctive. The `innerHeight` and `outerHeight` properties of the `window` JavaScript object, mentioned by [84, 102] but without any distinctiveness measure, have a respective normalized entropy of 0.388 and 0.327. The size of bounding boxes was used by [35] as a method of font fingerprinting. From the entropy reported in [35], we obtain a normalized entropy of 0.761 against 0.369 in our dataset. To the best of our knowledge, no previous study use the width and the position of a newly created `div` element as a fingerprinting attribute. However, it is highly distinctive as it achieves a normalized entropy of 0.324 in our dataset. All the mentioned attributes provide a sameness rate above 90%, except for the size of bounding boxes that goes down to 47%. However, when looking at its extracted parts, we observe that they have a sameness rate higher than 90% at the exception of the height of the first bounding box that has a sameness rate of 49%. This illustrates the necessity to break down some attributes to parts, as removing this part from the original attribute would drastically increase its sameness rate.

5.1.3 Attributes sameness rate. Figure 12 displays the cumulative distribution of the sameness rate among the attributes. Only 6 attributes (2.29%) provide a sameness rate below 85%, 5.7% of the attributes provide a sameness rate comprised in $[85, 95]$%, 10.7% provide a sameness rate comprised in $[95, 99]$%, and more than 80% of the attributes have a sameness rate above 99%. Among the $3,725,373$ pairs of consecutive fingerprints, only 5 attributes never change (i.e., they show a sameness rate of 100%). They comprise whether cookies are enabled, and four non-standard HTTP headers [50] that are `X-ATT-DeviceId`, `X-UIDH`, `X-WAP-Profile`, and `X-Network-Info`. These headers are typically added by mobile network operators to the communications of mobile devices [100]. The cookie enabling state is stable because the working dataset only contains fingerprints that have this functionality activated (see Section 2.4). As for the non-standard headers, the two first headers are always missing from the headers. The `X-Network-Info` header has three non-missing values that were seen on a single browser each. The `X-WAP-Profile` has three values that were seen on 37 browsers and consist of URL links directing to an XML document that describes the device. The most common link is shared by 35 browsers and concerns the Samsung B550H mobile phone [83].

The attributes that have a sameness rate below 85% are the size of bounding boxes, three extracted attributes derived from the bounding boxes, and two other attributes that we describe here. The instability of the size of bounding boxes is already explained in Section 5.1.2. The `Cache-Control` HTTP header allows the browser to specify the cache policy used during requests. It is the second

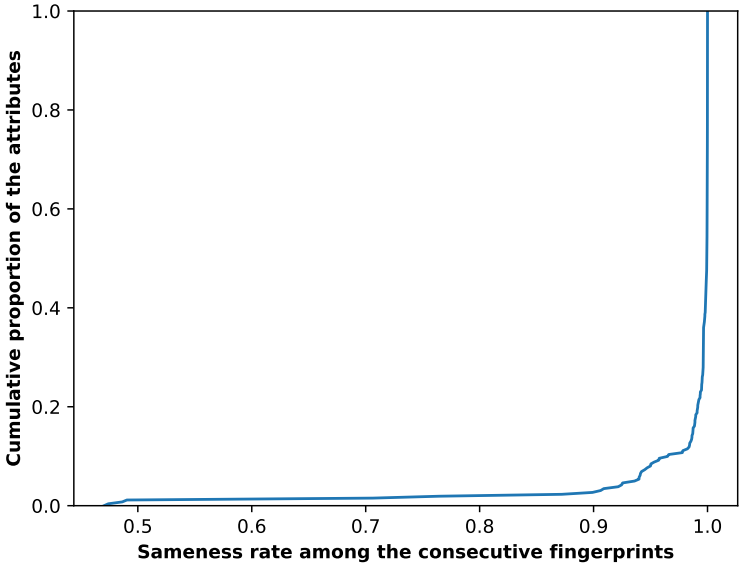

Fig. 12. Cumulative distribution of the sameness rate of the attributes among the consecutive fingerprints.

most instable attribute with a sameness rate of 70.63%. This header is instable because it is not always sent and some of its values contain the `max-age` parameter that can vary between requests. The third most instable attribute is the WebRTC fingerprinting method that has a sameness rate of 76.46%. This is due to three factors. First, the experimental state of this attribute and the bad support of the WebRTC API at the time of the experiment [5, 10] results in it being unsupported, undefined, or erroneous for 75.23% of the observed fingerprints. Then, it contains local IP addresses[10] which can change between two observations. Finally, it is composed by numerous pieces of information about an instance of a WebRTC connection, hence the change of any information modifies the value of the whole attribute. We reduce this effect by collecting these pieces of information from two connections and only keeping the identical values.

5.1.4 Attributes collection time. Most of the attributes are HTTP headers or JavaScript properties that are collected sequentially and in a negligible amount of time. We call *asynchronous attributes* the attributes that are collected asynchronously (i.e., in parallel). Only 33 attributes have a median collection time (MCT) higher than 5ms. They are presented in Figure 13 and are thoroughly described in Appendix A. All these attributes are asynchronous at the exception of the canvases, hence the total collection time of the fingerprint is not their sum. These attributes can be separated in three classes: extension detection, browser component, and media related. Their high collection time can be explained by these attributes waiting for the web page to render, executing heavy processes (e.g., graphical computation), or being asynchronous attributes that are affected by the side effect described in Appendix B. We consider the fingerprints that take more than 30 seconds to collect as outliers. We configured our fingerprinting script to collect the attributes in a given amount of time, after which the fingerprint is sent without waiting for all the attributes. In such case, the missing attribute do not have a collection time, and we ignore their collection time in the presented results. For all the attributes except five asynchronous attributes, these two types of

[10]Our script hashes these local IP addresses in MD5 directly on the clients, see [92] for more information about how the WebRTC API gives access to this information.

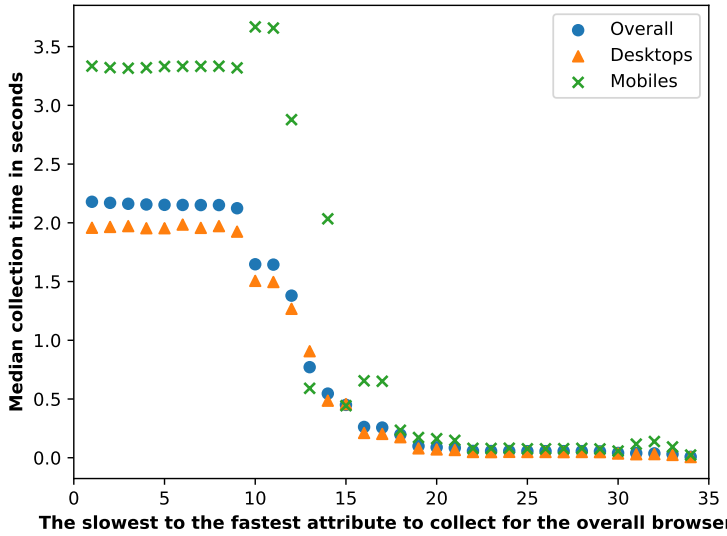

Fig. 13. Median collection time in seconds of the 33 attributes that have a median collection time higher than 5ms. The attributes are ranked from the slowest to the fastest to collect for the overall browsers. We consider the fingerprints that take more than 30 seconds to collect as outliers. The attributes that were not collected in time are not counted in these results. These two types of outliers account for less than 1% of the overall, desktop, and mobile entries, at the exception of the five asynchronous attributes discussed in Appendix B.

outliers account for less than 1% of the overall, desktop, and mobile entries. Appendix B discusses these two types of outliers.

The first class of the attributes that take time to collect are the methods of *extension detection*. All these attributes are asynchronous. The 9 slowest attributes detect an extension by the changes it brings to the content of the web page [89]. They have a median collection time (MCT) of approximately 2.2 seconds due to the waiting time before checking the changes on the web page. There is a clear difference between desktop and mobile browsers that show a respective MCT of approximately 2 seconds against 3.3 seconds. The 22nd to the 29th slowest attributes detect an extension based on web-accessible resources [86], which consist in checking whether an image embarked in an extension is accessible or not. They have a MCT of around 56ms, which is lower than the method that relies on detecting changes brought to the web page.

The second class of the attributes that take time to collect infer the availability of *browser components*. All these attributes are asynchronous. The list of speech synthesis voices is ranked 14th with a MCT of 546ms. This is due to this attribute being a list and the collection method that, for some browsers, requires to be done during an `onvoiceschanged` event which takes time to be triggered. The list of fonts and the inference of the default font are ranked 15th and 19th with a respective MCT of 450ms and 99ms. This is due to the detection method that measures the size of newly created text boxes [35]. The size of bounding boxes, the browser component colors, and the width and position of a newly created `div` element are ranked 18th, 20th, and 21st, with a MCT ranging from 84ms to 197ms. This is due to their waiting for and manipulation of the web page that takes time. The WebRTC fingerprinting method is ranked 13th with a MCT of 771ms. This is due to the creation of two WebRTC connections that are needed to gather information about the WebRTC configuration.

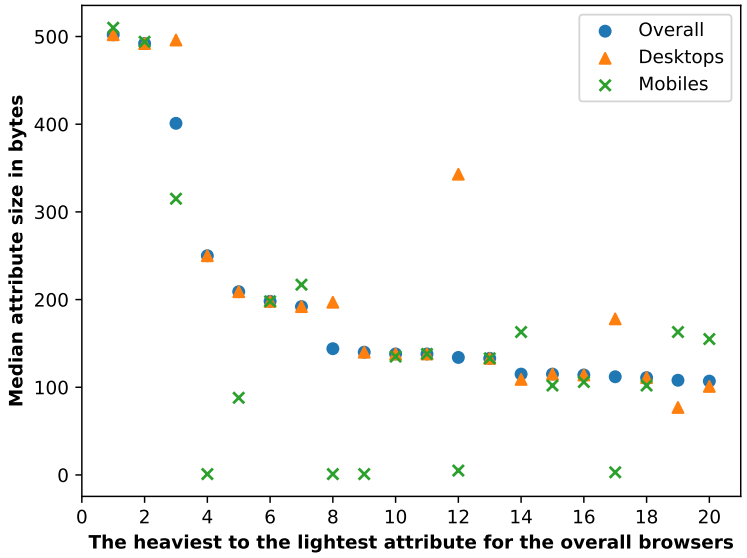

Fig. 14. Median size of the 20 heaviest attributes in bytes. The attributes are ranked from the heaviest to the lightest for the overall browsers.

The third class of the attributes that take time to collect are those that generate a media file (e.g., a sound, an image) and are discussed in Section 5.3. The methods of advanced audio fingerprinting are asynchronous and ranked 10th, 11th, and 12th. Our designed canvases are ranked 16th and 17th. The canvases inspired by the AmIUnique study [60] are ranked 31st and 33rd. The canvas similar to the canvas of the Morellian study [56] is ranked 32nd.

5.1.5 Attributes size. Most of our attributes have a negligible median size (MS). We have 137 attributes with a MS below 5 bytes, 105 attributes with a MS between 5 bytes and 100 bytes, and 20 attributes with a MS above 100 bytes. Figure 14 displays the median size of the 20 heaviest attributes in bytes, ranked from the heaviest to the lightest attribute for the overall browsers. We discuss below these 20 heaviest attributes.

The heaviest attributes are composed of *list attributes* and *verbose textual attributes*. The three heaviest attributes are list attributes. The list of the properties of the `navigator` object is the heaviest attribute with a MS of 502 bytes, the list of colors of layout components is second with a MS of 492 bytes, and the list of WebGL extensions is third with a MS of 401 bytes. Examples of verbose textual attributes are the `appVersion` that is 18th with a MS of 107 bytes, and the `userAgent` JavaScript property that is 14th with a MS of 115 bytes, which is more verbose than its HTTP header counterpart that is 19th with a MS of 108 bytes.

On mobile browsers, some list attributes are most of the time empty due to their lack of customization (e.g., plugins are mostly unsupported). They are the list of speech synthesis voices that is 4th, the list of the constraints supported by the `mediaDevices` object that is 8th, the list of plugins that is 12th, and the list of supported mime types that is 17th. On the contrary, the verbose attributes are slightly heavier on mobile browsers, which is explained by the presence of additional information like the device model.

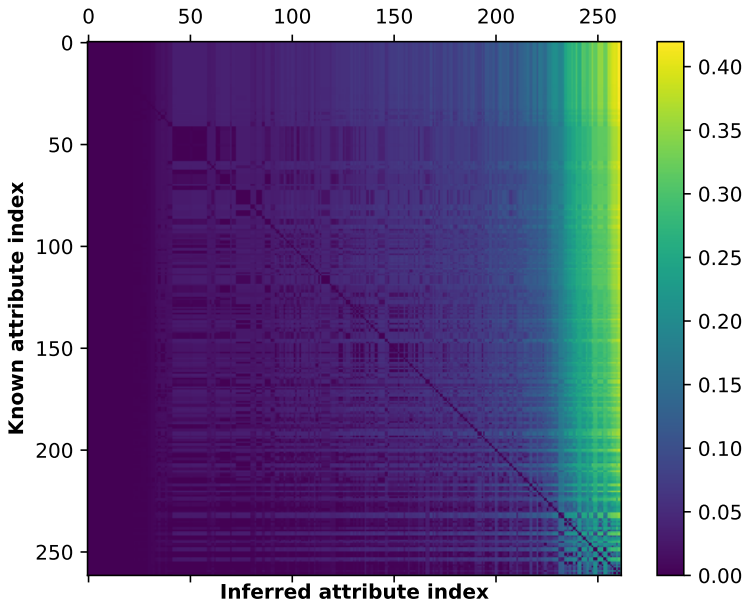

Fig. 15. The normalized conditional entropy (NCE) of the attributes given the knowledge of another attribute. Given the known attribute of index j on the vertical axis, the remaining NCE of the inferred attribute of index i on the horizontal axis is read in the cell (i, j). The attributes are ordered by the average normalized conditional entropy and the indices match with Figure 16.

5.2 Correlation between attributes

We can expect to have correlations occurring between the attributes when considering more than 200 attributes. We provide here an overview of the correlation between the attributes, that include the nine dynamic attributes, and refer the reader to Appendix E for insight in the correlation of each attribute.

For comparability with the results of attributes distinctiveness, we express the correlation by the conditional entropy $\mathrm{H}(a_j|a_i)$ of an attribute a_j when another attribute a_i is known, normalized to the maximum entropy H_M. We call this measure the *normalized conditional entropy*. It is comprised between 0.0 if knowing a_i allows to completely infer a_j, and the normalized entropy of a_j if knowing a_i provides no information on the value of a_j (i.e., they are independent). We denote V_i the domain of the attribute a_i and e_v^i the event that the attribute a_i takes the value v. We consider the relative frequency p of the attribute values among the considered fingerprints. The measure of the conditional entropy $\mathrm{H}(a_j|a_i)$ of a_j given a_i is expressed as

$$\mathrm{H}(a_j|a_i) = -\sum_{v \in V_i, w \in V_j} p(e_v^i, e_w^j) \log \frac{p(e_v^i, e_w^j)}{p(e_v^i)} \tag{9}$$

Figure 15 displays the normalized conditional entropy (NCE) of the attributes given the knowledge of another attribute and Figure 16 displays the minimum, the average, and the maximum NCE of an attribute when the value of another attribute is known. In these two figures, the attributes are ordered identically by the average NCE. We recall that the maximum entropy is $H_M = 21.983$ bits and that a conditional entropy of 1 bit is equivalent to a NCE of 0.045.

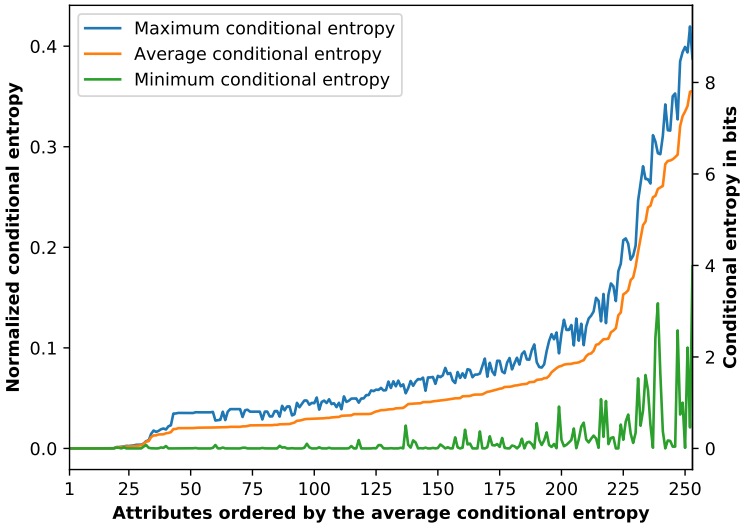

Fig. 16. Minimum, average, and maximum normalized conditional entropy of an attribute when the value of another attribute is known. The attributes are ordered by the average normalized conditional entropy and the indices match with Figure 15.

In Figure 16, we ignore the 9 source attributes from which the extracted attributes are derived and the comparison of an attribute with itself. These cases are irrelevant as the extracted attributes are completely correlated with their source attribute, and an attribute is completely correlated with itself. We obtain a total of 253 attributes. We stress that the maximum NCE of an attribute is always equal to the normalized entropy of this attribute. This is due to the cookie enabling state that is always `true`. As a result, it provides a null entropy and knowing its value does not provide any information on the value of the other attributes. We can see three parts in this figure. First, 19 attributes have a low normalized entropy of less than 10^{-3} and their NCE when knowing another attribute is at most as much. Few different values have been observed for these attributes. Then, 194 attributes have an average NCE between 10^{-3} and 10^{-1}. The minimum normalized conditional entropy (MinNCE) of these attributes is near 0.0, hence there exists another attribute that can be used to efficiently infer their value. Finally, 40 attributes have an average NCE higher than 10^{-1} and generally have a strictly positive MinNCE. These attributes help to efficiently distinguish browsers and are less correlated to other attributes.

The minimum normalized conditional entropy (MinNCE) is an interesting indicator of the efficiency to infer the value of an attribute if the value of another attribute is known. We have 49 attributes that have a null MinNCE and can completely be inferred when another attribute is known. Moreover, 192 attributes have a MinNCE comprised in the range $]0, 0.045]$, hence knowing the value of another attribute helps to infer their value but not completely. Finally, 12 attributes have a MinNCE higher than 0.045, which is equivalent to having a minimum conditional entropy higher than 1 bit. They are displayed in Table 5. They consist of highly distinctive attributes that concern the screen or window size (e.g., W.screenX, W.innerHeight), browser components (e.g., the list of fonts or plugins), and verbose information about the browser (e.g., the UserAgent) or about external components (e.g., WebRTC fingerprinting).

As the attributes tend to be correlated with each other, a subset of them can provide sufficient distinctiveness or the same distinctiveness as when using all the attributes. Several attribute

Table 5. The attributes that have a minimum normalized conditional entropy (MinNCE) higher than 0.045, which is equivalent to a minimum conditional entropy higher than 1 bit. The minimum conditional entropy is in bits. W refers to the `window` JavaScript object, WG refers to an initialized WebGL context, and [...] denotes a truncated part.

Attribute	MinNCE	Minimum conditional entropy (in bits)
W.innerHeight	0.181	3.98
WebRTC fingerprinting	0.144	3.17
W.outerHeight	0.117	2.58
List of fonts	0.110	2.42
List of plugins	0.100	2.21
W.outerWidth	0.074	1.63
WebGL renderer (unmasked)	0.073	1.61
Height of first bounding box	0.070	1.53
S.availHeight	0.058	1.27
W.screenY	0.049	1.08
W.screenX	0.047	1.04
userAgent JavaScript property	0.046	1.00

selection methods were proposed by previous works. The most common methods weight the attributes (e.g., by the entropy) and successively pick the attribute of the highest weight until the obtained set reaches a threshold (e.g., on the number of attributes) [24, 46, 54, 70, 93]. Other selection methods leverage the correlation that occur between the attributes and pick the next attribute according to the attributes that are already selected [15, 35, 77]. In particular, the authors of the FPSelect study [15] apply three attribute selection methods to a sample of our dataset and compare the methods according to the distinctiveness and the usability of the resulting fingerprints.

5.3 Focus on dynamic attributes

Previous studies highlight the possibility to fingerprint browsers by rendering media files inside the browser, given instructions that are provided by the fingerprinter (e.g., WebGL canvas [71], HTML5 canvas [28], audio fingerprinting [78]). Later on, these attributes were integrated within challenge-response mechanisms [56, 80] that mitigate replay attacks. We call these attributes the *dynamic attributes* and include nine of them in our script: five HTML5 canvases, three audio fingerprinting methods, and a WebGL canvas. To the best of our knowledge, no study evaluates the properties of several dynamic attributes on a browser population, together with the evaluation of various set of instructions. In this section, we seek to fill this gap and focus on the properties offered by the nine dynamic attributes of our fingerprinting script.

5.3.1 HTML5 canvas. The HTML5 canvas consists in asking the browser to draw an image using the canvas API within the two-dimensional context called `2d`. This method is already studied in several works, whether it is about its efficacy [28, 44, 60], its use on the web [34, 61], or the distinctiveness provided by various sets of instructions [56]. However, to the best of our knowledge, no study evaluates the different properties (e.g., distinctiveness, stability, collection time) offered by various sets of complex instructions (i.e., mixes of texts, emojis, mathematical curves, drawn using different colors). We seek to fill this gap and evaluate the properties of five HTML5 canvases generated following three sets of instructions and in two image formats (PNG and JPEG). We name the canvases given the set of instructions and the format. We call *AmIUnique canvas* the canvas

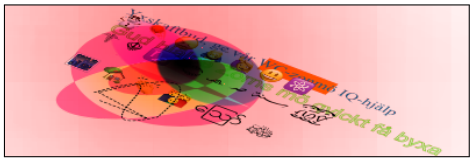

Fig. 17. A sample of our designed HTML5 canvas in PNG format.

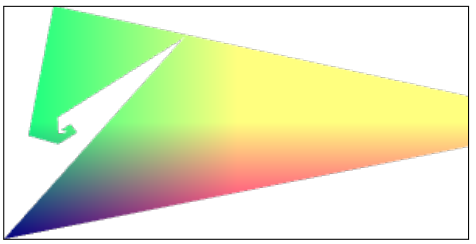

Fig. 18. A sample of our designed WebGL canvas in PNG format.

generated by the set of instructions inspired by the AmIUnique[11] study [60]. We extract it in PNG format only. We call *Morellian canvas* the canvas generated by the set of instructions that is similar to the Morellian study [56]. We extract it in PNG and JPEG format. We call *custom canvas* the canvas generated by a set of instructions that we designed. We extract it in PNG and JPEG format. We generate the canvas using the API, then we collect the base64 representation of the image in PNG and JPEG format using the `toDataURL` function. The `quality` parameter of this function goes from 0.0 to 1.0 and allow us to control the level of compression of the JPEG versions. As we seek to compare the PNG canvases that are compressed without loss with their JPEG counterparts using a high level of compression, we set the `quality` to 0.1 for the JPEG versions. An example of the custom canvas in PNG format is displayed in Figure 17. Examples of the AmIUnique and Morellian canvas are provided in Appendix A.

We observe that canvases are less distinctive and more stable in JPEG format than in PNG format. For example, the custom canvas has a normalized entropy of 0.420 when exported in PNG against 0.399 for the JPEG version. However, the PNG version has a sameness rate of 92.16% against 93.59% for the JPEG version. These differences are due to distinct images in PNG format ending up the same after the lossy compression of the JPEG format. Acar et al. [9] consider that a canvas generated in a lossy compression format as JPEG is not an indicator of a fingerprinting attempt. Although the JPEG version provides a lower distinctiveness than the PNG version, it is still highly distinctive when generated using a complex set of instructions. Indeed, it is the second most distinctive attribute among ours (see Section 5.1.2). The time overhead induced by the additional extraction in JPEG format is negligible. For example, the custom canvas has a median collection time (MCT) increased by 5ms for the JPEG version compared to the PNG version that has a MCT of 257ms. We do not account the size differences as both formats are hashed and the resulting hashes weigh 64 bytes. The PNG and the JPEG versions are also highly correlated, as knowing the value in the PNG format of the custom canvas leaves a normalized conditional entropy (NCE) of 5.28×10^{-4} on the value of the JPEG format. On the opposite, knowing the value of the JPEG format provides a higher NCE of 0.021 on the value of the PNG format.

The properties of the three PNG canvases differ, with the custom canvas being the most distinctive. First, the Morellian canvas is an enhanced version of the AmIUnique canvas with additional curves. This enhancement provides an increase of the distinctiveness, with a normalized entropy of 0.385 for the Morellian canvas against 0.353 for the AmIUnique canvas. However, it comes with a loss of stability with a respective sameness rate of 94.71% against 98.64%. Then, the custom canvas is more complex than the Morellian canvas as it includes several emojis, two strings including Swedish letters, many overlapping ellipses, a background with a color gradient, and a rotation of all these

[11] Contrarily to the AmIUnique study [60], we only have one sentence and one smiling face emoji instead of two. Also, we do not draw a colored rectangle and the sentence is drawn using a color gradient.

elements if the functionality is available. These improvements provide a higher normalized entropy of 0.420, but a lower sameness rate of 92.16%. The main drawback of adding more complexity to the custom canvas is the temporal cost. It has a MCT of 257ms against 37ms for the Morellian canvas and 31ms for the AmIUnique canvas. However, it represents less than 10% of the total median collection time of the fingerprints which is 2.92 seconds, and less than 5% of the median loading time of a web page on a desktop browser which is 6.5 seconds [19]. Finally, knowing the value of the custom canvas leaves less variability on the value of the two other canvases than the opposite. When knowing the value of the Morellian or AmIUnique canvas, the custom canvas has a respective NCE of 0.079 and 0.103. However, knowing the value of the custom canvas results in the Morellian canvas showing a NCE of 0.044 and the AmIUnique canvas showing a NCE of 0.037. To conclude, adding more instructions for the canvas drawing usually provides more distinctiveness, as each instruction can induce a difference between browsers. However, it comes at the cost of additional computation time and loss of stability as each additional instruction can constitute an instability factor.

5.3.2 WebGL canvas. The WebGL canvas is an image that is drawn using the canvas API within the `webgl` or `webgl2` contexts. These contexts use the WebGL library [8] that leverages hardware accelerations to render and manipulate two-dimensional graphics and three-dimensional scenes. Canvas fingerprinting was first introduced by Mowery et al. [71] using the `webgl` context, but afterward most studies focused on the `2d` context [10, 22, 28, 101]. This can result from the unreliability of the WebGL context encountered by Laperdrix et al. [60] for which Cao et al. [29] proposed a remedy by setting specific parameters.

Our WebGL canvas consists in sequential triangles with a color gradient and is simpler than our designed HTML5 canvas. It provides a normalized entropy of 0.263 against 0.420 for the custom HTML5 canvas. It is more stable that the other canvases with a sameness rate of 98.97%, and has a median collection time of 41ms against 257ms for the custom HTML5 canvas. Figure 18 displays an example of our WebGL canvas.

5.3.3 Web Audio. Web Audio fingerprinting was discovered by Englehardt et al. [34] when assessing the use of web tracking methods on the web, and was studied thoroughly by Queiroz et al. [78]. It consists in processing audio signals in the browser and collecting the rendered result. Similarly to canvases, this processing relies on software and hardware components, and variations occur between different component stacks. It works by creating an `Audio Context`, which in our case is the `OfflineAudioContext`[12], in which we manipulate `Audio Nodes`. The `Audio Nodes` are of three types. Source nodes generate an audio signal (e.g., from a microphone or an audio stream), destination nodes render the signal (e.g., playing it through speakers), and manipulation nodes manipulate the signal (e.g., increase its volume). These nodes are linked together to form a network that goes from source nodes to destination nodes and passes through manipulation nodes. We refer the reader to [78] for a broader description of the main audio nodes.

We have three audio fingerprinting attributes. The *audio FP simple* (AFS) attribute relies on a simple process. The attributes *audio FP advanced* (AFA) and *audio FP advanced frequency data* (AFA-FD) rely on a more advanced process. Their concrete implementation is described in Appendix A. The most distinctive audio attribute is the AFA-FD that has a normalized entropy of 0.161, followed by the AFS (0.153), and the AFA (0.147). They all have a sameness rate of approximately 95%. Their values are the string representations of floating-point numbers, hence they have a median size of 17 bytes. The simple process has a median collection time (MCT) of 1.38 seconds and the advanced

[12]The advantage of using OfflineAudioContext is the manipulation of audio signal without playing any sound on a genuine audio output peripheral.

process has a MCT of 1.64 seconds. The AFA-FD is collected from the advanced version and has a MCT increased by 3ms compared to the AFA.

6 DISCUSSION

This section describes how browser fingerprinting can be integrated in a web authentication mechanism and discusses the attacks that are possible on such mechanism.

6.1 Browser fingerprinting-based authentication mechanism

Browser fingerprinting can enhance an authentication mechanism by providing an additional barrier at a low usability and deployability cost. In this section, we provide an example of how it can concretely be implemented. The verifier controls a web platform on which her users are registered. Each registered account is associated to an identifier and to a set of authentication factors. They include a simple password and the fingerprints of the browsers used by the account owner. We emphasize that dynamic attributes can be integrated to the browser fingerprints to enforce a challenge-response mechanism that mitigates replay attacks [56, 80].

6.1.1 Enrollment. The enrollment consists for a user to create his account and to register the authentication factors. During this step, the user and the verifier agrees on the account identifier (e.g., username, email address, phone number) and on the authentication factors (e.g., password, email address, phone number) that are assigned to the user. The fingerprint of the browser in use during the enrollment is collected and stored as the first browser fingerprint of the user. To register an additional browser, the user is required to authenticate using other strong factors (e.g., a physical token, a one time password) before getting the fingerprint of the new browser registered.

6.1.2 Authentication. During each authentication, the user claims an account by providing the identifier and presenting the authentication factors. The verifier compares the presented authentication factors with the ones stored for this user. If they match, the user is deemed legitimate and is given access to the account. Otherwise, the access is denied to the user and the verifier can take preventive actions [42] according to her policy (e.g., blocking the account). The verifier notably compares the collected browser fingerprint with the fingerprints of the browsers registered to the account. If the other factors match and the fingerprint of one of the registered browser matches the collected fingerprint, the fingerprint stored for this browser is updated to the newly collected one. By using the `User-Agent` HTTP header, it would also be possible to reduce the set of the stored fingerprints against which to compare the presented fingerprint (e.g., recognizing the family of the browser in use and getting the stored fingerprints of this family).

6.1.3 Account Recovery. It happens that legitimate users are not able to provide the authentication factors (e.g., a password is forgotten, a physical token is lost). When it occurs, users are given access to an account recovery mechanism [68] that leverages other authentication factors than the usual ones (e.g., face-to-face verification, email verification). Users cannot mistake their browser fingerprint, but it can become hard to recognize if too many changes are brought to the web environment of the browser. In this case, the user is asked to undergo the account recovery step and select the registered browser for which to update its fingerprint.

6.2 Attacks on the authentication mechanism

We discuss in this section the attacks that are possible on the browser fingerprinting-based authentication mechanism. We consider an attacker that can access the authentication page and seeks to impersonate as many legitimate users as possible. To impersonate a legitimate user, the attacker has

to find the right combination of identifier and authentication factors. Below, we make assumptions about the attacker and describe the attacks that he can execute.

6.2.1 Assumptions about the attacker. We focus on the contribution of browser fingerprinting to the authentication mechanism and make the following assumptions. The attacker knows the identifiers of the targeted users and the authentication factors (e.g., their password) at the exception to the browser fingerprint[13]. A concrete example is an attacker that obtained these pieces of information from a data leak or theft. The attacker cannot tamper with the authentication server of the verifier nor with the device or the browser of the user. The attacker cannot tamper nor eavesdrop the communications between the server of the verifier and the browser of the user. The attacker knows the attributes that are collected by the fingerprinting script. This knowledge can be obtained from static or dynamic analysis of the script [20, 23, 79]. The attacker knows the domains of the attributes and the domain of the fingerprints. The latter is the Cartesian product of the former. This knowledge can stem from publicly accessible resources like statistics [95], documentation, or fingerprint datasets [97, 101]. It can also come from malicious sources like stolen fingerprints [69], phishing attacks [96], or a set of controlled browsers [75]. The attacker can control the values of the attributes that compose the fingerprint by using open source tools like Disguised Chromium Browser [22] or Blink [59], commercial solutions like Multilogin [63] or AntiDetect [18], or by altering the network packets that contain the fingerprint with tools like BurpSuite [64]. The attacker can submit a limited number of arbitrary fingerprints to try to impersonate a targeted user. We emphasize that the reach of the attacker depends on the number of attempts similarly to guessing attacks on passwords [25, 104]. In our online authentication context, the number of attempts depends on the rate limiting policy of the verifier [42, 65]. After a given number of failed attempts, the verifier can lock the account and ask the account owner to update his authentication factors (e.g., change his password, update his fingerprint).

6.2.2 Brute force attack. The brute force attack consists in the attacker submitting randomly sampled fingerprints from the fingerprint domain. In practice, the attacker randomly picks the value of each attribute, forges a fingerprint by combining them, and submits this fingerprint. The attacker repeats this process until the account is locked out or after a number of attempts decided by herself. The reach of the brute force attack depends on the space of the possible fingerprints that the attacker manages to cover in the limited number of attempts. Two factors greatly limit this reach. First, the domain of the fingerprint grows exponentially with the domain of the composing attributes. Second, the fingerprints are sampled randomly from the possibility space, which can lead to highly improbable combination of attribute values due to contradicting or incompatible values. Contradicting values occur when two attributes that always provide the same value, like the `logicalXDPI` and `logicalYDPI` properties of the `screen` JavaScript object, are chosen with diverging values. Incompatible values occur when two attributes that are correlated are chosen with incompatible values, like one value that indicates a mobile device (e.g., a UserAgent typically observed on mobile devices) and the other value indicating a desktop device (e.g., a large screen size). On our experimental setup, these two factors are important and greatly reduce the reach of brute force attacks. Indeed, our dataset is composed of 262 attributes (including the extracted ones) among which 20% have more than a thousand distinct values (see Section 5). Moreover, our attributes tend to be highly correlated with each other: 49 attributes have a null conditional entropy when the value of another attribute is known (see Section 5.2). Although the fingerprint verification mechanism can allow differences between the fingerprints, a randomly sampled fingerprint is

[13]We discuss in Section 6.2.4 the case where all the authentication factors are known to the attacker including the browser fingerprint.

expected to have a lower number of identical attributes compared to the fingerprint of the targeted user. For a given attribute, there is only one chance over the size of the attribute domain for the attribute to be identical, and there may be more chances for it to be similar. However, for the randomly sampled fingerprint to match the target fingerprint, there should be sufficient identical or similar attributes, whereas the chances for this to happen decreases with the number and domain size of the attributes.

6.2.3 Dictionary attack. The dictionary attack consists for an attacker to infer a distribution of the fingerprints (e.g., from stolen fingerprints [69]) and to submit the most probable fingerprints. The reach of the dictionary attack depends on the correspondence between the fingerprint distribution among the protected users and the fingerprint distribution known by the attacker. We take the example of an attacker having the knowledge of a fingerprint distribution obtained from mobile browsers configured in English, but targeting users of desktop browsers configured in French. In this example, the fingerprints submitted by the attacker would have few chances to match as some attributes (e.g., the screen size, the browser language) would differ. The users that are impersonated are those for which one of the registered fingerprints matches with one of the most probable fingerprints given the knowledge of the attacker. The authors of the FPSelect study [15] measured the reach of a dictionary attack on a sample of 30,000 fingerprints of our dataset, considering an attacker that knows the exact distribution of the target fingerprints and a verification mechanism that allows differences between the compared fingerprints. They found out that 0.21% of the users are impersonated when the attacker is able to submit the 4 most common fingerprints, which goes up to 0.51% for the 16 most common fingerprints. These proportions are a higher bound on the proportion of the fingerprints that are shared respectively by the 4 and the 16 most common fingerprints due to the differences allowed by the verification mechanism. These proportions are slightly lower than the share of the most common passwords measured in previous studies. Thomas et al. [96] analyzed nearly two billion passwords and found out that the 4 most common passwords are shared by 0.72% of the users, which goes up to 1.01% for the 10 most common passwords. Wang et al. [104] worked on nine password datasets and found out that the ten most common passwords are shared by 0.79% to 10.44% of the users, with an average of 3.06% among the datasets.

6.2.4 Replay attack. The replay attack consists for an attacker to collect the fingerprint of a user (e.g., through a phishing attack) and to submit this fingerprint to impersonate the user. Recent works [56, 80] proposed to use dynamic attributes – notably the HTML5 canvas – to design challenge-response mechanisms to thwart replay attacks. These challenge-response mechanisms leverage the value of the dynamic attributes which depends on the set of instructions that is used (e.g., the drawing instructions of the canvas). The challenge is then the set of instructions and the response is the value generated by the browser. By varying the challenge on each authentication, a fingerprint that would have been collected by the attacker would be invalid as it would be the response to a previous challenge. We refer to the work of Laperdrix et al. [56] for a thorough analysis of the level of security provided by such challenge-response mechanism. Section 5.3 analyzes the nine dynamic attributes of three different types that are included in our study.

6.2.5 Relay attack. The relay attack consists for an attacker to pose as a legitimate user to the verifier, to pose as the verifier to a targeted user, and to relay the communications from one to the other. A relay attack is divided in the three following steps. First, the attacker sets a phishing website that embarks an authentication page similar to the verifier. Then, the attacker lures a victim to visit this fake page. At the same time, the attacker connects to the authentication page of the verifier and forwards the fingerprinting script to the victim. The victim enters her credentials and gets her browser fingerprint collected. Finally, the attacker submits these pieces of information to

the authentication page of the verifier. Amnesty International reports the usage of relay attacks in real life and shows that relay attacks are able to break two-factor authentication [48]. Laperdrix et al. [56] acknowledge that a challenge-response mechanism is theoretically vulnerable to relay attacks. However, they discuss the practical implications of relay attacks and explain that relay attacks are costly for the attackers to execute in practice. Outside the web context, distance bounding protocols [21] were proposed to defend against relay attacks [38]. We let the design of a distance bounding protocol that leverages the time taken by the browser and the server to communicate as a future work.

7 RELATED WORKS

In this section, we present related works about the use of browser fingerprinting for authentication. In addition, Section 2.5 compares our dataset with the previously studied large-scale datasets [33, 44, 60, 62, 77] and Section 5 compares the distinctiveness of our attributes with the distinctiveness reported in previous studies.

We stress that most studies about browser fingerprinting focus on their use for identification. The use of browser fingerprinting for identification and for authentication differ in the objective when given a presented fingerprint f. In identification, we seek to find the identity (e.g., an account, an advertisement profile) to which f belongs among a pool of N candidate identities. If f is unrecognized, it is associated to a new identity that is then added to the candidate identities. Hence, in identification we operate a 1 to N comparison. Depending on the matching method, the matching time can be proportional to N and become unrealistic in a large-scale setting [62]. On the contrary, in authentication the identity is already given (e.g., the claimed account) and we seek to verify that f legitimately belongs to this identity. Hence, in authentication we operate a 1 to 1 comparison[14]. The studies about the use of browser fingerprinting for identification usually evaluate the threat posed to privacy by browser accessible information [33, 35, 44, 60, 71, 78, 89], propose counter-measures to avoid being tracked by this technique [22, 57], or measure its usage on the web [9, 23, 34, 49].

The first works about the use of browser fingerprinting for authentication focused on its use for continuous authentication [76, 88, 99]. Their objective is to verify that the authenticated session is not hijacked. These studies focus on the integration of browser fingerprinting in an authentication mechanism and provide few insights about the properties of the fingerprints (e.g., only [88] analyzes fingerprints and focuses on their classification efficacy). On the contrary, we identify several properties of authentication factors with which we evaluate real-life browser fingerprints, and explain the results by highlighting the contribution of single attributes. Moreover, these studies only consider a small fraction of the hundreds of available attributes and do not include the dynamic attributes that can be used in a challenge-response mechanism to thwart replay attacks. We include more than 200 attributes – including 9 dynamic attributes – for which we provide the implementation and the properties (e.g., number of distinct values, normalized entropy).

Alaca et al. [13] provided a classification of the fingerprinting attributes according to properties that include the stability, the distinctiveness, and the resource usage. They qualitatively estimate these properties given the nature of the attributes. For some attributes, they provide the entropy measured in previous studies and acknowledge that further study is needed on this subject. We stress that comparing the attributes entropy between datasets of different sizes leads to comparability problems as explained by [60] (e.g., the attribute of a dataset of N fingerprints provides at most

[14]If users register n browsers to their account, as described in Section 6.1, f is compared to the fingerprint of these n browsers that are already identified. Users are expected to register fewer than ten devices [82], hence n is expected to be smaller than N (e.g., a website having a thousand accounts registered). It would also be possible to find the right fingerprint among the n possibilities by leveraging the UserAgent to recognize which browser the user is using.

an entropy of $\log_2(N)$ bits). We provide the quantitative measure of these properties on our attributes from the analysis of real-life fingerprints, and also evaluate the accuracy of a simple verification mechanism. About the attributes considered by [13], they include attributes that require interactions from the user (e.g., her location) or that are related to the network protocol (e.g., the TCP/IP stack fingerprinting that analyzes the response to specially crafted messages). Moreover, their classification leads to different attributes being grouped under a single designation, like the "major software and hardware details" that includes the attribute family of the JavaScript properties that we describe in Appendix A. The attributes of this family can show diverse properties, like the UserAgent that provides a normalized entropy of 0.394 and a sameness rate of 0.98%, whereas the `screenX` property of the `window` object provides a normalized entropy of 0.125 and a sameness rate of 0.93%. We show that more than 200 attributes are easily accessible (i.e., they require few lines of JavaScript) and do not require any interaction from the user, which would reduce the usability (e.g., the user being asked permissions several times). Moreover, we provide the exhaustive list of the attributes with their concrete implementation and their properties. We also evaluate the properties of distinctiveness, stability, and resource usage on the complete fingerprints that combine as many attributes.

Markert et al. [68] recently presented a *work in progress* about the long-term analysis of fallback authentication methods. They plan to measure the recovery rate of the evaluated methods after an elapsed time of 6, 12, and 18 months. These methods include browser fingerprinting, for which they acknowledge that "not much about browser fingerprinting-based security systems is known".

Rochet et al. [80] and Laperdrix et al. [56] proposed challenge-response mechanisms that rely on dynamic attributes, especially the HTML5 canvas. The instructions provided to the canvas API are changed on each authentication, making the generated image vary on each fingerprinting. Trivial replay attacks then fails as the awaited canvas image changes each time. In this study, we propose the evaluation of nine dynamic attributes that include five HTML5 canvases, one WebGL canvas, and three audio fingerprinting methods.

8 CONCLUSION

In this study, we conduct the first large-scale empirical study of the properties of browser fingerprints when used for web authentication. We make the link between the digital fingerprints that distinguish Humans, and the browser fingerprints that distinguish browsers, to evaluate the latter according to properties inspired by biometric authentication factors. We formalize and evaluate the properties for the browser fingerprints to be usable and practical in a web authentication context. They include the distinctiveness of the fingerprints, their stability, their collection time, their size, the loss of efficacy among browser types, and the accuracy of a simple illustrative verification mechanism. We evaluate these properties on a real-life large-scale fingerprint dataset collected over a period of 6 months that contains 4, 145, 408 fingerprints composed of 216 attributes. The attributes include nine dynamic attributes which are used in state-of-the-art authentication mechanisms to mitigate replay attacks. We thoroughly describe the preprocessing steps to prepare the dataset and the browser population that, contrary to most of the previous studies, is not biased towards technically-savvy users. We show that, considering time-partitioned datasets, our browser fingerprints provide a unicity rate above 81.3% which is stable over the 6 months, and more than 94.7% of the fingerprints are shared by at most 8 browsers. We observe a loss of distinctiveness from mobile browsers that show a lower unicity rate of 42%. About the stability and on average, more than 91% of the attributes are identical between two observations of the fingerprint of a browser, even when they are separated by nearly 6 months. About the memory and the time consumption of our fingerprints, we show that they weigh a dozen of kilobytes and take a few seconds to collect. Our simple verification mechanism achieves an equal error rate of 0.61%. It comes from most of the consecutive fingerprints of a browser

having at least 234 identical attributes, whereas most of the fingerprints of different browsers have fewer. To better comprehend the results on the complete fingerprints, we evaluate the contribution of the attributes to each fingerprint property. We show that although the attributes show a lower distinctiveness compared to previous studies, 10% of the attributes provide a normalized entropy higher than 0.25. Moreover, four attributes missing from the previous large-scale studies are part of the most distinctive of our attributes. They concern the size and position of elements of the browser interface. About the stability, 85% of the attributes stay identical between 99% of the pairs of consecutive fingerprints coming from the same browser. Few attributes consume a high amount of resources. Only 33 attributes take more than 5ms to collect and only 20 attributes weigh more than 100 bytes. When looking at the correlation between the attributes, we find that 49 attributes can completely be inferred when the value of another attribute is known. We remark that the dynamic attributes are part of the most time-consuming attributes, but also among the most distinctive attributes. We show the importance of the set of instructions which influences both the distinctiveness and the stability of the generated values (e.g., the canvas image). We conclude that the browser fingerprints obtained from the combination of the studied browser population, and the large surface of fingerprinting attributes, carry the promise to strengthen web authentication mechanisms.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments. We also thank Benoît Baudry and David Gross-Amblard for their valuable comments.

REFERENCES

[1] 2002. *Consolidated text: Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector (Directive on privacy and electronic communications).* https://data.europa.eu/eli/dir/2002/58/2009-12-19 accessed on 2021-06-21.

[2] 2009. *Consolidated text: Directive 2009/136/EC of the European Parliament and of the Council of 25 November 2009 amending Directive 2002/22/EC on universal service and users' rights relating to electronic communications networks and services, Directive 2002/58/EC concerning the processing of personal data and the protection of privacy in the electronic communications sector and Regulation (EC) No 2006/2004 on cooperation between national authorities responsible for the enforcement of consumer protection laws (Text with EEA relevance).* https://data.europa.eu/eli/dir/2009/136/2009-12-19 accessed on 2021-06-21.

[3] 2017. *Browser Market Share France | StatCounter Global Stats.* https://gs.statcounter.com/browser-market-share/all/france/2017 accessed on 2021-06-21.

[4] 2017. *Operating System Market Share France | StatCounter Global Stats.* https://gs.statcounter.com/os-market-share/all/france/2017 accessed on 2021-06-21.

[5] 2021. *"createDataChannel" | Can I use... Support tables for HTML5, CSS3, etc.* https://caniuse.com/?search=createDataChannel accessed on 2021-06-21.

[6] 2021. *"SpeechSynthesis" | Can I use... Support tables for HTML5, CSS3, etc.* https://caniuse.com/?search=SpeechSynthesis accessed on 2021-06-21.

[7] 2021. *"Web Audio API" | Can I use... Support tables for HTML5, CSS3, etc.* https://caniuse.com/?search=WebAudioAPI accessed on 2021-06-21.

[8] 2021. *WebGL.* https://get.webgl.org accessed on 2021-06-21.

[9] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2014). 674–689. https://doi.org/10.1145/2660267.2660347

[10] Nasser Mohammed Al-Fannah and Wanpeng Li. 2017. Not All Browsers are Created Equal: Comparing Web Browser Fingerprintability. In *International Workshop on Security (IWSEC)* (2017), Satoshi Obana and Koji Chida (Eds.). 105–120. https://doi.org/10.1007/978-3-319-64200-0_7

[11] Nasser Mohammed Al-Fannah, Wanpeng Li, and Chris J. Mitchell. 2018. Beyond Cookie Monster Amnesia: Real World Persistent Online Tracking. In *Information Security*, Liqun Chen, Mark Manulis, and Steve Schneider (Eds.). Springer, 481–501. https://doi.org/10.1007/978-3-319-99136-8_26

[12] Nasser Mohammed Al-Fannah and Chris Mitchell. 2020. Too Little Too Late: Can We Control Browser Fingerprinting? 21, 2 (2020), 165–180. https://doi.org/10.1108/JIC-04-2019-0067

[13] Furkan Alaca and P. C. van Oorschot. 2016. Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods. In *Annual Conference on Computer Security Applications (ACSAC)* (2016-10). 289–301. https://doi.org/10.1145/2991079.2991091

[14] Inc. Alexa Internet. 2021. *Top Sites in France - Alexa.* https://www.alexa.com/topsites/countries/FR accessed on 2021-06-21.

[15] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2020. FPSelect: Low-Cost Browser Fingerprints for Mitigating Dictionary Attacks against Web Authentication Mechanisms. In *Annual Computer Security Applications Conference (ACSAC)* (2020). https://doi.org/10.1145/3427228.3427297

[16] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2021. "Guess Who?" Large-Scale Data-Centric Study of the Adequacy of Browser Fingerprints for Web Authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (2021), Leonard Barolli, Aneta Poniszewska-Maranda, and Hyunhee Park (Eds.). 161–172. https://doi.org/10.1007/978-3-030-50399-4_16

[17] Tompoariniaina Nampoina Andriamilanto. 2020. Leveraging Browser Fingerprinting for Web Authentication. https://tel.archives-ouvertes.fr/tel-03150590

[18] Antidetect. 2021. *Antidetect.* https://antidetect.org accessed on 2021-06-21.

[19] The HTTP Archive. 2020. *Median Loading Time of Web Pages.* https://httparchive.org/reports/loading-speed#ol accessed on 2021-06-21.

[20] Mohammadreza Ashouri. 2019. A Large-Scale Analysis of Browser Fingerprinting via Chrome Instrumentation. 25–36. https://www.thinkmind.org/index.php?view=article&articleid=icimp_2019_2_20_30045

[21] Gildas Avoine, Muhammed Ali Bingöl, Ioana Boureanu, Srdjan Čapkun, Gerhard Hancke, Süleyman Kardaş, Chong Hee Kim, Cédric Lauradoux, Benjamin Martin, Jorge Munilla, Alberto Peinado, Kasper B. Rasmussen, Dave Singelée, Aslan Tchamkerten, Rolando Trujillo-Rasua, and Serge Vaudenay. 2019. Security of Distance-bounding: A Survey. 51, 5 (2019), 1–33. https://doi.org/10.1145/3264628

[22] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. 2016. Disguised Chromium Browser: Robust Browser, Flash and Canvas Fingerprinting Protection. In *ACM Workshop on Privacy in the Electronic Society (WPES)* (2016). 37–46. https://doi.org/10.1145/2994620.2994621

[23] Sarah Bird, Vikas Mishra, Steven Englehardt, Rob Willoughby, David Zeber, Walter Rudametkin, and Martin Lopatka. 2020. Actions Speak Louder than Words: Semi-supervised Learning for Browser Fingerprinting Detection. (2020). https://arxiv.org/abs/2003.04463

[24] C. Blakemore, J. Redol, and M. Correia. 2016. Fingerprinting for Web Applications: From Devices to Related Groups. In *IEEE Trustcom/BigDataSE/ISPA* (2016-08). 144–151. https://doi.org/10.1109/TrustCom.2016.0057

[25] Joseph Bonneau. 2012. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *IEEE Symposium on Security and Privacy (S&P)* (2012-05). 538–552. https://doi.org/10.1109/SP.2012.49

[26] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. 2015. Passwords and the Evolution of Imperfect Authentication. 58, 7 (2015), 78–87. https://doi.org/10.1145/2699390

[27] Ralph Broenink. 2012. Using Browser Properties for Fingerprinting Purposes. In *Twente Student Conference on IT* (2012).

[28] Elie Bursztein, Artem Malyshev, Tadek Pietraszek, and Kurt Thomas. 2016. Picasso: Lightweight Device Class Fingerprinting for Web Clients. In *Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)* (2016-10-24). 93–102. https://doi.org/10.1145/2994459.2994467

[29] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *Network and Distributed System Security Symposium (NDSS)* (2017-01-01). https://doi.org/10.14722/ndss.2017.23152

[30] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2018). 1515–1532. https://doi.org/10.1145/3243734.3243860

[31] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and Xiaofeng Wang. 2014. The Tangled Web of Password Reuse. In *Network and Distributed System Security Symposium (NDSS)* (2014). 23–26. https://doi.org/10.14722/ndss.2014.23357

[32] Data Is Beautiful. 2019. Usage Share of Internet Browsers 1996 - 2019. https://www.youtube.com/watch?v=es9DNe0l0Qo accessed on 2021-06-21.

[33] Peter Eckersley. 2010. How Unique is Your Web Browser?. In *International Conference on Privacy Enhancing Technologies (PETS)* (2010). 1–18. https://doi.org/10.1007/978-3-642-14527-8_1

[34] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2016-10-24). 1388–1401. https://doi.org/10.1145/2976749.2978313

[35] David Fifield and Serge Egelman. 2015. Fingerprinting Web Users Through Font Metrics. In *Financial Cryptography and Data Security (FC)* (2015), Rainer Böhme and Tatsuaki Okamoto (Eds.). 107–124. https://doi.org/10.1007/978-3-662-47854-7_7

[36] fingerprintjs. 2021. *fingerprintjs/fingerprintjs: Browser fingerprinting library with the highest accuracy and stability.* https://github.com/fingerprintjs/fingerprintjs accessed on 2021-06-21.

[37] Micro Focus. 2019. Device Fingerprinting for Low Friction Authentication. https://www.microfocus.com/media/white-paper/device-fingerprinting-for-low-friction-authentication-wp.pdf accessed on 2021-06-21.

[38] Aurélien Francillon, Boris Danev, and Srdjan Capkun. 2011. Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars. In *Network and Distributed System Security Symposium (NDSS)* (2011). https://www.ndss-symposium.org/wp-content/uploads/2017/09/franc.pdf

[39] Marco Gamassi, Massimo Lazzaroni, Mauro Misino, Vincenzo Piuri, Daniele Sana, and Fabio Scotti. 2005. Quality Assessment of Biometric Systems: A Comprehensive Perspective based on Accuracy and Performance Measurement. 54, 4 (2005), 1489–1496. https://doi.org/10.1109/TIM.2005.851087

[40] Ewa Gasperowicz. 2018. *OffscreenCanvas — Speed up Your Canvas Operations with a Web Worker.* https://developers.google.com/web/updates/2018/08/offscreen-canvas accessed on 2021-06-21.

[41] Tom Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. 2016. Accelerometer-Based Device Fingerprinting for Multi-factor Mobile Authentication. In *International Symposium on Engineering Secure Software and Systems (ESSoS)* (2016). 106–121. https://doi.org/10.1007/978-3-319-30806-7_7

[42] Maximilian Golla, Theodor Schnitzler, and Markus Dürmuth. 2018. "Will Any Password Do?" Exploring Rate-Limiting on the Web. In *USENIX Symposium on Usable Privacy and Security (SOUPS)* (2018-08-12).

[43] Google. 2017. *Background Tabs in Chrome 57 | Web | Google Developers.* https://developers.google.com/web/updates/2017/03/background_tabs#background_timer_alignment accessed on 2021-06-21.

[44] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *The Web Conference (TheWebConf)* (2018-04). https://doi.org/10.1145/3178876.3186097

[45] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support Vector Machines. 13, 4 (1998), 18–28. https://doi.org/10.1109/5254.708428

[46] Peter Hraška. 2018. Browser Fingerprinting. https://virpo.sk/browser-fingerprinting-hraska-diploma-thesis.pdf

[47] Troy Hunt. 2018. *Troy Hunt: 86% of Passwords are Terrible (and Other Statistics).* https://www.troyhunt.com/86-of-passwords-are-terrible-and-other-statistics accessed on 2021-06-21.

[48] Amnesty International. 2018. *When Best Practice Isn't Good Enough: Large Campaigns of Phishing Attacks in Middle East and North Africa Target Privacy-Conscious Users.* https://www.amnesty.org/en/latest/research/2018/12/when-best-practice-is-not-good-enough accessed on 2021-06-21.

[49] U. Iqbal, S. Englehardt, and Z. Shafiq. 2021. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *IEEE Symposium on Security and Privacy (S&P)* (2021-05). IEEE Computer Society, 283–301. https://doi.org/10.1109/SP40001.2021.00017

[50] jonarne. 2008. *Useful "X headers" - mobiForge.* https://mobiforge.com/design-development/useful-x-headers accessed on 2021-06-21.

[51] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing* (2 ed.). Pearson. 23–27 pages.

[52] Nian-hua KANG, Ming-zhi CHEN, Ying-yan FENG, Wei-ning LIN, Chuan-bao LIU, and Guang-yao LI. 2017. Zero-Permission Mobile Device Identification Based on the Similarity of Browser Fingerprints. In *International Conference on Computer Science and Technology (CST)* (2017-07-31). https://doi.org/10.12783/dtcse/cst2017/12531

[53] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *Network and Distributed System Security Symposium (NDSS)* (2020). https://doi.org/10.14722/ndss.2020.24383

[54] Amin Faiz Khademi, Mohammad Zulkernine, and Komminist Weldemariam. 2015. An Empirical Evaluation of Web-Based Fingerprinting. 32, 4 (2015), 46–52. https://doi.org/10.1109/MS.2015.77

[55] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. 2016. Fingerprinting Mobile Devices Using Personalized Configurations. 2016, 1 (2016). https://doi.org/10.1515/popets-2015-0027

[56] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. 2019. Morellian Analysis for Browsers: Making Web Authentication Stronger With Canvas Fingerprinting. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2019-06). 43–66. https://doi.org/10.1007/978-3-030-22038-9_3

[57] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing Core Browser Objects to Break Advanced Device Fingerprinting Techniques. In *International Symposium on Engineering Secure Software and Systems (ESSoS)* (2017), Eric Bodden, Mathias Payer, and Elias Athanasopoulos (Eds.). 97–114. https://doi.org/10.1007/978-3-319-62105-0_7

[58] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting: A Survey. 14, 2 (2020), 8:1–8:33. https://doi.org/10.1145/3386040

[59] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2015. Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification. In *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2015-05). 98–108. https://doi.org/10.1109/SEAMS.2015.18

[60] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *IEEE Symposium on Security and Privacy (S&P)* (2016-05). 878–894. https://doi.org/10.1109/SP.2016.57

[61] H. Le, F. Fallace, and P. Barlet-Ros. 2017. Towards Accurate Detection of Obfuscated Web Tracking. In *IEEE International Workshop on Measurement and Networking (M&N)* (2017-09). 1–6. https://doi.org/10.1109/IWMN.2017.8078365

[62] Song Li and Yinzhi Cao. 2020. Who Touched My Browser Fingerprint? A Large-Scale Measurement Study and Classification of Fingerprint Dynamics. In *ACM Internet Measurement Conference* (2020) *(IMC '20)*. Association for Computing Machinery, 370–385. https://doi.org/10.1145/3419394.3423614

[63] Multilogin Software Ltd. 2021. *Multilogin - Replace Multiple Computers With Virtual Browser Profiles - Multilogin.* https://multilogin.com accessed on 2021-06-21.

[64] PortSwigger Ltd. 2021. *Burp Suite - Application Security Testing Software - PortSwigger.* https://portswigger.net/burp accessed on 2021-06-21.

[65] Bo Lu, Xiaokuan Zhang, Ziman Ling, Yinqian Zhang, and Zhiqiang Lin. 2018. A Measurement Study of Authentication Rate-Limiting Mechanisms of Modern Websites. In *Annual Computer Security Applications Conference (ACSAC)* (2018-12-03). 89–100. https://doi.org/10.1145/3274694.3274714

[66] Davide Maltoni, Dario Maio, Anil K. Jain, and Salil Prabhakar. 2009. *Handbook of Fingerprint Recognition* (2 ed.). Springer. 8–24 pages. https://doi.org/10.1007/978-1-84882-254-2

[67] Francesco Marcantoni, Michalis Diamantaris, Sotiris Ioannidis, and Jason Polakis. 2019. A Large-scale Study on the Risks of the HTML5 WebAPI for Mobile Sensor-based Attacks. In *The Web Conference (TheWebConf)* (2019). 3063–3071. https://doi.org/10.1145/3308558.3313539

[68] Philipp Markert, Maximilian Golla, Elizabeth Stobert, and Markus Dürmuth. 2020. Work in Progress: A Comparative Long-Term Study of Fallback Authentication. In *Network and Distributed System Security Symposium (NDSS)* (2020). https://www.ndss-symposium.org/ndss-paper/auto-draft-30/

[69] Paul Marks. 2020. Dark Web's Doppelgängers Aim to Dupe Antifraud Systems. 63, 2 (2020), 16–18. https://doi.org/10.1145/3374878

[70] João Pedro Figueiredo Correia Rijo Mendes. 2011. noPhish – Anti-phishing System using Browser Fingerprinting. https://estagios.dei.uc.pt/cursos/mei/relatorios-de-estagio/?id=279

[71] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. (2012), 1–12. https://www.ieee-security.org/TC/W2SP/2012/papers/w2sp12-final4.pdf

[72] Mozilla. 2021. *Service Worker API - Web APIs | MDN.* https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API accessed on 2021-06-21.

[73] Mozilla. 2021. *WindowOrWorkerGlobalScope.setTimeout() - Web APIs | MDN.* https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout accessed on 2021-06-21.

[74] Mozilla and individual contributors. 2021. *NavigatorPlugins.plugins - Web APIs | MDN.* https://developer.mozilla.org/en-US/docs/Web/API/NavigatorPlugins/plugins accessed on 2021-06-21.

[75] Panagiotis Papadopoulos, Panagiotis Ilia, Michalis Polychronakis, Evangelos P. Markatos, Sotiris Ioannidis, and Giorgos Vasiliadis. 2019. Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation. In *Network and Distributed System Security Symposium (NDSS)* (2019-02). https://doi.org/10.14722/ndss.2019.23070

[76] Davy Preuveneers and Wouter Joosen. 2015. SmartAuth: Dynamic Context Fingerprinting for Continuous User Authentication. In *Annual ACM Symposium on Applied Computing (SAC)* (2015). 2185–2191. https://doi.org/10.1145/2695664.2695908

[77] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. 2020. Long-Term Observation on Browser Fingerprinting: Users' Trackability and Perspective. 2020, 2 (2020), 558–577. https://doi.org/10.2478/popets-2020-0041

[78] Jordan S. Queiroz and Eduardo L. Feitosa. 2019. A Web Browser Fingerprinting Method Based on the Web Audio API. (2019). https://doi.org/10.1093/comjnl/bxy146

[79] Valentino Rizzo, Stefano Traverso, and Marco Mellia. 2021. Unveiling Web Fingerprinting in the Wild Via Code Mining and Machine Learning. 2021, 1 (2021), 43 – 63. https://doi.org/10.2478/popets-2021-0004

[80] Florentin Rochet, Kyriakos Efthymiadis, François Koeune, and Olivier Pereira. 2019. SWAT: Seamless Web Authentication Technology. In *The Web Conference (TheWebConf)* (2019-05). 1579–1589. https://doi.org/10.1145/3308558.3313637

[81] Julian F. Reschke Roy T. Fielding. 2014. *RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* https://tools.ietf.org/html/rfc7231#section-5.5.3 accessed on 2021-06-21.

[82] Bardia Safaei, Amir Mahdi Monazzah, Milad Bafroei, and Alireza Ejlali. 2017. Reliability Side-Effects in Internet of Things Application Layer Protocols. https://doi.org/10.1109/ICSRS.2017.8272822

[83] Samsung. 2015. *SAMSUNG UMTS Handset UA Prof.* http://wap.samsungmobile.com/uaprof/SM-B550H.xml accessed on 2021-06-21.

[84] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In *Network and Distributed System Security Symposium (NDSS)* (2019). https://doi.org/10.14722/ndss.2019.23155

[85] SecureAuth. 2020. *Device / Browser Fingerprinting - Heuristic-based Authentication.* https://docs.secureauth.com/pages/viewpage.action?pageId=33063454 accessed on 2021-06-21.

[86] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. In *ACM Conference on Data and Application Security and Privacy (CODASPY)* (2017). 329–336. https://doi.org/10.1145/3029806.3029820

[87] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2015. Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication. In *European Workshop on System Security (EuroSec)* (2015). 6:1–6:6. https://doi.org/10.1145/2751323.2751329

[88] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2017. Leveraging Battery Usage from Mobile Devices for Active Authentication. (2017). https://doi.org/10.1155/2017/1367064

[89] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *IEEE Symposium on Security & Privacy (S&P)* (2017-05-24). 941–956. https://doi.org/10.1109/SP.2017.18

[90] StatCounter. 2017. *Browser Market Share Worldwide | StatCounter Global Stats.* https://gs.statcounter.com/browser-market-share/all/worldwide/2017 accessed on 2021-06-21.

[91] StatCounter. 2017. *Operating System Market Share Worldwide | StatCounter Global Stats.* https://gs.statcounter.com/os-market-share/all/worldwide/2017 accessed on 2021-06-21.

[92] K. Takasu, T. Saito, T. Yamada, and T. Ishikawa. 2015. A Survey of Hardware Features in Modern Browsers: 2015 Edition. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (2015-07). 520–524. https://doi.org/10.1109/IMIS.2015.72

[93] Kazuhisa Tanabe, Ryohei Hosoya, and Takamichi Saito. 2018. Combining Features in Browser Fingerprinting. In *Advances on Broadband and Wireless Computing, Communication and Applications (BWCCA)* (2018), Leonard Barolli, Fang-Yie Leu, Tomoya Enokido, and Hsing-Chung Chen (Eds.). 671–681. https://doi.org/10.1007/978-3-030-02613-4_60

[94] Adobe Communications Team. 2017. *Flash & The Future of Interactive Content.* https://blog.adobe.com/en/publish/2017/07/25/adobe-flash-update.html accessed on 2021-06-21.

[95] The Carat Team. 2021. *Carat Project Statistics.* http://carat.cs.helsinki.fi/statistics accessed on 2021-06-21.

[96] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. 2017. Data Breaches, Phishing, or Malware? Understanding the Risks of Stolen Credentials. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2017-10-30). 1421–1434. https://doi.org/10.1145/3133956.3134067

[97] Henning Tillmann. 2014. *Browser Fingerprinting: 93% of all user configurations are unique | Henning Tillmann.* https://www.henning-tillmann.de/en/2014/05/browser-fingerprinting-93-of-all-user-configurations-are-unique accessed on 2021-06-21.

[98] Christof Ferreira Torres, Hugo Jonker, and Sjouke Mauw. 2015. FP-Block: Usable Web Privacy by Controlling Browser Fingerprinting. In *European Symposium on Research in Computer Security (ESORICS)* (2015-09-21). 3–19. https://doi.org/10.1007/978-3-319-24177-7_1

[99] T. Unger, M. Mulazzani, D. Frühwirt, M. Huber, S. Schrittwieser, and E. Weippl. 2013. SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting. In *International Conference on Availability, Reliability and Security (ARES)* (2013-09). 255–261. https://doi.org/10.1109/ARES.2013.33

[100] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, and Vern Paxson. 2015. Header Enrichment or ISP Enrichment?: Emerging Privacy Threats in Mobile Networks. In *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization (HotMiddlebox)* (2015). 25–30. https://doi.org/10.1145/2785989.2786002

[101] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *IEEE Symposium on Security and Privacy (S&P)* (2018-05-21). 728–741. https://doi.org/10.1109/sp.2018.00008

[102] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. 2020. FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)* (2020).

[103] Rick Waldron. 2021. *Generic Sensor API.* https://www.w3.org/TR/2021/CRD-generic-sensor-20210619 accessed on 2021-06-21.

[104] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. 2016. Targeted Online Password Guessing: An Underestimated Threat. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2016-10-24). 1242–1254. https://doi.org/10.1145/2976749.2978339

[105] Stephan Wiefling, Luigi Lo Iacono, and Markus Dürmuth. 2019. Is This Really You? An Empirical Study on Risk-Based Authentication Applied in the Wild. In *IFIP International Conference on ICT Systems Security and Privacy Protection (SEC)* (2019). 134–148. https://doi.org/10.1007/978-3-030-22312-0_10

[106] Wenjia Wu, Jianan Wu, Yanhao Wang, Zhen Ling, and Ming Yang. 2016. Efficient Fingerprinting-Based Android Device Identification with Zero-Permission Identifiers. 4 (2016), 8073–8083. https://doi.org/10.1109/ACCESS.2016.2626395

[107] Vasilios Zorkadis and P. Donos. 2004. On Biometrics-based Authentication and Identification from a Privacy-protection Perspective: Deriving Privacy-enhancing Requirements. 12, 1 (2004), 125–137. https://doi.org/10.1108/09685220410518883

A BROWSER FINGERPRINTING ATTRIBUTES

In this section, we describe the 216 *base attributes* that are included in our script and the 46 *extracted attributes* that are derived from the 9 source attributes. We group the attributes in families and provide references to related studies. Their name is sufficient to retrieve the corresponding browser property. When needed, we provide a brief description of the method for reproducibility. We focus here on the description of the method and provide a *complete list* of the attributes and their property in Appendix E. We denote `A.[B, C]` a property that is accessed either through `A.B` or `A.C`. If there is only one element inside the brackets, this element is optional. We denote `[...]` a part that is omitted but described in the corresponding attribute description.

A.1 JavaScript properties

Most attributes are *properties* that are accessed through common *JavaScript objects*. The `navigator` object provides information on the browser (e.g., its version), its customization (e.g., the configured language), the underlying system (e.g., the operating system), and the supported functionalities (e.g., the list of available codecs). The `screen` object provides information on the screen size, the orientation, the pixel density, and the available space for the web page. The `window` object provides information on the window containing the web page, like its size or the support of storage mechanisms. The `document` object gives access to the page content and a few properties. The JavaScript properties are already included in previous studies [33, 44, 60] or open source projects [36], but are usually limited to fewer than 20 properties.

A.2 HTTP headers

Our script collects 16 attributes from *HTTP headers*, most of which are already used in previous studies [33, 44, 60]. Among these 16 attributes, 15 consist of the value of an explicitly specified header and the last attribute stores any remaining fields as pairs of name and value.

A.3 Enumeration or presence of browser components

One attribute family that provides a high diversity is the *browser components*. The presence of some components can directly be accessed (e.g., the list of plugins [15]) whereas the presence of others have to be inferred (e.g., the installed fonts). The list of components that are given in this section have the components separated by a comma.

A.3.1 List attributes. Previous studies already identified the *list of plugins* and the *list of fonts* as highly distinctive [33, 60]. We include these two attributes in our fingerprinting script. We

[15]Firefox browsers since version 29 require developers to probe the exact name of the wanted plugin and do not allow them to enumerate the list of plugins [74]. This version was released in April 29, 2014, hence the Firefox browsers of our population are impacted by this limitation. As a result, the list of plugins of our Firefox browsers mostly contains a single entry that concerns the Flash plugin.

enumerate the *list of plugins* and the *list of mime types* (i.e., the supported data format). The list of fonts is an asynchronous attribute. To collect it, we check the size of text boxes [35] to infer the presence of 66 fonts. Additionally, we collect the *list of speech synthesis voices* from the `speechSynthesis` property of the `window` object or one of its variant (e.g., prefixed with `webkit`). To do so, we use the `getVoices` function of this property. On some browsers, we have to wait for the `onvoiceschanged` event to be triggered before getting access to the list. This attribute is then asynchronous.

A.3.2 Support of video, audio, and streaming codecs. We infer the *support of video codecs* by creating a `video` element and checking whether it can play a given type using the `canPlayType()` function. The *support of audio codecs* is inferred by applying the same method on an `audio` element. We infer the *support of streaming codecs* by calling the `isTypeSupported()` function of the `window.[WebKit, moz, ms, ∅]MediaSource` object and checking the presence of both the audio and the video codecs. We apply the same method to infer the *support of recording codecs* from the `MediaRecorder` object.

A.3.3 List of video codecs. The 15 *video codecs* for which we infer the presence are the following: video/mp2t; codecs="avc1.42E01E,mp4a.40.2", video/mp4; codecs="avc1.42c00d", video/mp4; codecs="avc1.4D401E", video/mp4; codecs="mp4v.20.8", video/mp4; codecs="avc1.42E01E", video/mp4; codecs="avc1.42E01E, mp4a.40.2", video/mp4; codecs="hvc1.1.L0.0", video/mp4; codecs="hev1.1.L0.0", video/ogg; codecs="theora", video/ogg; codecs="vorbis", video/webm; codecs="vp8", video/webm; codecs="vp9", application/dash+xml, application/vnd.apple.mpegURL, audio/mpegurl.

A.3.4 List of audio codecs. The 9 *audio codecs* for which we infer the presence are the following: audio/wav; codecs="1", audio/mpeg, audio/mp4; codecs="mp4a.40.2", audio/mp4; codecs="ac-3", audio/mp4; codecs="ec-3", audio/ogg; codecs="vorbis", audio/ogg; codecs="opus", audio/webm; codecs="vorbis", audio/webm; codecs="opus".

A.3.5 List of detected fonts. The 66 *fonts* for which we infer the presence are the following: Andale Mono; AppleGothic; Arial; Arial Black; Arial Hebrew; Arial MT; Arial Narrow; Arial Rounded MT Bold; Arial Unicode MS; Bitstream Vera Sans Mono; Book Antiqua; Bookman Old Style; Calibri; Cambria; Cambria Math; Century; Century Gothic; Century Schoolbook; Comic Sans; Comic Sans MS; Consolas; Courier; Courier New; Garamond; Geneva; Georgia; Helvetica; Helvetica Neue; Impact; Lucida Bright; Lucida Calligraphy; Lucida Console; Lucida Fax; LUCIDA GRANDE; Lucida Handwriting; Lucida Sans; Lucida Sans Typewriter; Lucida Sans Unicode; Microsoft Sans Serif; Monaco; Monotype Corsiva; MS Gothic; MS Outlook; MS PGothic; MS Reference Sans Serif; MS Sans Serif; MS Serif; MYRIAD; MYRIAD PRO; Palatino; Palatino Linotype; Segoe Print; Segoe Script; Segoe UI; Segoe UI Light; Segoe UI Semibold; Segoe UI Symbol; Tahoma; Times; Times New Roman; Times New Roman PS; Trebuchet MS; Verdana; Wingdings; Wingdings 2; Wingdings 3.

A.4 Extension detection

The list of the installed *browser extensions* cannot be directly accessed, but their presence can be inferred. We check the changes that are brought to the web page [89] by the 8 extensions that are listed in Table 6 and the availability of the web accessible resources [53, 86] of the 8 extensions that are listed in Table 7. All these 16 attributes are asynchronous.

A.4.1 Detection of an ad-blocker. We infer the presence of an *ad-blocker* by creating an invisible dummy advertisement and checking whether it is removed or not. This attribute is then asynchronous. To detect the removal, we check whether the common methods used by ad-blockers are applied. They consist into removing the DOM element, setting its size to zero (e.g., using the `offsetWidth` or `clientHeight` property), setting its `visibility` to `hidden`, or setting its

Table 6. Extensions detected by the changes they bring to the page content.

Extension	Page content change
Privowny	W.privownyAddedListener[EXT] is supported
UBlock	D.head has `display: none !important;` and `:root` as style
Pinterest	D.body.data-pinterest-extension-installed is supported
Grammarly	D.body.data-gr-c-s-loaded is supported
Adguard	W.AG_onLoad is supported
Evernote	Element with `style-1-cropbar-clipper` as `id` exists
TOTL	W.ytCinema is supported
IE Tab	W.ietab.getVersion() is supported

Table 7. Extensions detected by the availability of their web accessible resource. C stands for chrome and R stands for resource.

Extension	Web accessible resource
Firebug	C://firebug/skin/firebugBig.png
YahooToolbar	R://635abd67-4fe9-1b23-4f01-e679fa7484c1/icon.png
EasyScreenshot	C://easyscreenshot/skin/icon16.png
Ghostery	R://firefox-at-ghostery-dot-com/data/images/ghosty-16px.png
Kaspersky	R://urla-at-kaspersky-dot-com/data/icon-16.png
VideoDownloadHelper	R://b9db16a4-6edc-47ec-a1f4-b86292ed211d/data/images/icon-18.png
GTranslate	R://aff87fa2-a58e-4edd-b852-0a20203c1e17/icon.png
Privowny	C://privowny/content/icons/privowny_extension_logo.png

`display` to `none`. The dummy advertisement is a created division which has the `id` property set to "`ad_ads_pub_track`", the `class` set to "`ads .html?ad= /?view=ad text-ad textAd text_ad text_ads text-ads`", and the `style` set to "`width: 1px !important; height: 1px !important; position: absolute !important; left: -1000px !important; top: -1000px !important;`".

A.5 Size and color of web page elements

A.5.1 Bounding boxes. The attributes related to the *bounding boxes* concern a `div` element to which we append a `span` element. These attributes are asynchronous. The `div` element has his `style` property set to the values displayed in Table 8. The `span` element contains a specifically crafted text that is provided below. The size of bounding boxes (i.e., the width and the height of the rectangles of the `div` and the `span` elements) are then collected using the `getClientRects` function. The text of the `span` element is "`\ua9c0 \u2603 \u20B9 \u2604 \u269b \u2624 \u23B7 \u262c \u2651 \u269d \u0601 \u0603 \uaac1 \u060e \u06dd \ud83c\udfe1 mmmmmmmmmmlil \u102a`". The characters that start with "\u" are special Unicode characters (e.g., emojis, letters of a non-latin alphabet).

A.5.2 Width and position of a created div. The attribute named *width and position* of a created `div` is the properties of `width` and `transform-origin` of a newly created `div` element which are obtained from the `getComputedStyle` function. This attribute is asynchronous. This created `div` element is afterward used to infer the color of layout components as described below.

Table 8. Properties of the `div` element that is measured to generate the value of the attributes related to the bounding boxes.

Property	Value
position	absolute
left	-9999px
textAlign	center
objectFit	scale-down
font	68px / 83px Helvetica, Arial, Sans-serif
zoom	66%
MozTransform	scale(0.66)
visibility	hidden

A.5.3 Colors of layout components. The attribute *colors of layout components* is obtained by applying the color of several layout components (e.g., the scroll bar) to the created `div` element denoted `new_div`, and getting the color back from the property `W.getComputedStyle(new_div).color`. Each of the tested component gets its color extracted this way and their colors are aggregated in this attribute. The color of each element is afterward extracted as a single attribute (see Section 2.4). They are displayed at the end of Table 16 that lists the attributes and their properties. These attributes are asynchronous.

A.6 WebGL properties

Our script collects several properties from the WebGL API. To obtain them, we create a `canvas` element and get its WebGL context by calling `getContext()` using any of the following parameters: `webgl`, `webgl2`, `moz-webgl`, `experimental-webgl`, or `experimental-webgl2`.

The property `MAX_TEXTURE_MAX_ANISOTROPY_EXT` is obtained from one of `[WEBKIT_EXT_, MOZ-_EXT_, EXT_]texture_filter_anisotropic`. To get the unmasked vendor and renderer, we first get an identifier named `id` from the unmasked property of the `getExtension('WEBGL_debug_renderer_info')` object, and then get the actual value by calling `getParameter(id)`. The unmasked property of the renderer is named `UNMASKED_RENDERER_WEBGL` and the one of the vendor is named `UNMASKED_VENDOR_WEBGL`. Finally, to get the `COMPRESSED_TEXTURE_FORMATS` property, we have to load the `[WEBKIT_]WEBGL_compressed_texture_s3tc` extension first.

A.7 WebRTC fingerprinting

We include a WebRTC fingerprinting method similar to the method proposed by Takasu et al. [92]. The method consists in getting information about the Session Description Protocol of a generated WebRTC connection. Due to the variability of this information, we create two different connections and hold only the values that are identical between them. As this method leaks local IP addresses, we hash them directly on the client before sending this attribute. This attribute is asynchronous.

A.8 HTML5 canvases inspired by previous studies

Our script includes a canvas inspired by the AmIUnique study [60] in both PNG and JPEG formats, and an enhanced version similar to the Morellian study [56]. Figure 19 and Figure 20 respectively display an example of a canvas generated using these set of instructions. We refer to Section 5.3 for a focus on the dynamic attributes.

Fig. 19. A sample of the HTML5 canvas that is inspired by the AmIUnique [60] study.

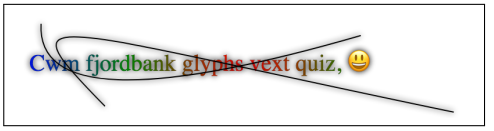

Fig. 20. A sample of the HTML5 canvas that is similar to the Morellian [56] study.

A.9 Audio fingerprinting

In this section, we provide the concrete implementation and the network of `AudioNode` objects used within each audio fingerprinting methods. These methods are inspired by the work of Englehardt et al. [34]. They are designed to form complex networks of `AudioNode` objects to have more chances to induce fingerprintable behaviors. The three audio fingerprinting attributes are asynchronous. We refer to Section 5.3 for a focus on the dynamic attributes.

A.9.1 Simple audio fingerprinting method. The *simple process* consists of three `OscillatorNode` objects that generate a periodic wave, connected to a single `DynamicsCompressorNode`, and finishing to a `AudioDestinationNode`. The architecture of the network of `AudioNode` objects for the simple process is depicted in Figure 21 with the parameters set for each node. The `OscillatorNode` objects are started one after the other and overlap at some time. The sequence of events is the following: (1) the triangle oscillator node is started at $t = 0$ seconds, (2) the square oscillator is started at $t = 0.10$ seconds, (3) the triangle oscillator is stopped at $t = 0.20$ seconds and the sine oscillator node is started, and finally (4) the square oscillator is stopped at $t = 0.25$ seconds. When the rendering of the audio context is done, the `complete` event is triggered and gives access to a `renderedBuffer` that contains the audio data encoded as 32 bits floating-point numbers. The *audio FP simple* (AFS) attribute is an integer computed as the sum of these numbers cast to absolute integers.

A.9.2 Advanced audio fingerprinting method. The *advanced process* consists of four `OscillatorNode` objects, two `BiquadFilterNode` objects, two `PannerNode` objects, one `DynamicsCompressorNode`, one `AnalyserNode`, and one `AudioDestinationNode`. The architecture of the network of `AudioNode` objects for the advanced process is depicted in Figure 22 with the parameters set for each node. The `OscillatorNode` objects are started one after the other and overlap at some time. The sequence of events is the following: (1) the triangle oscillator node and the sine oscillator node with a frequency of 280 are started at $t = 0$ seconds, (2) the square oscillator is started at $t = 0.05$ seconds, (3) the triangle oscillator is stopped at $t = 0.10$ seconds, (4) the sine oscillator with a frequency of 170 is stopped at $t = 0.15$ seconds, (5) the square oscillator is stopped at $t = 0.20$ seconds. When the rendering of the audio context is done, the `complete` event is triggered and gives access to a `renderedBuffer` that contains the audio data encoded as 32 bits floating-point numbers. The *audio FP advanced* (AFA) attribute is an integer computed as the sum of these numbers cast to absolute integers. The *audio FP advanced frequency data* (AFA-FD) attribute is the sum of the frequency data obtained through the `getFloatFrequencyData` function of the `AnalyserNode`.

B ANOMALOUS COLLECTION TIMES

In this section, we describe a side effect encountered by our script that results in some fingerprints and attributes showing a high collection time above 30 seconds. These high collection times concern the sending of the fingerprint and the collection of the attributes that require the body to be set or that are collected asynchronously. In total, 27 base attributes are impacted by these high collection

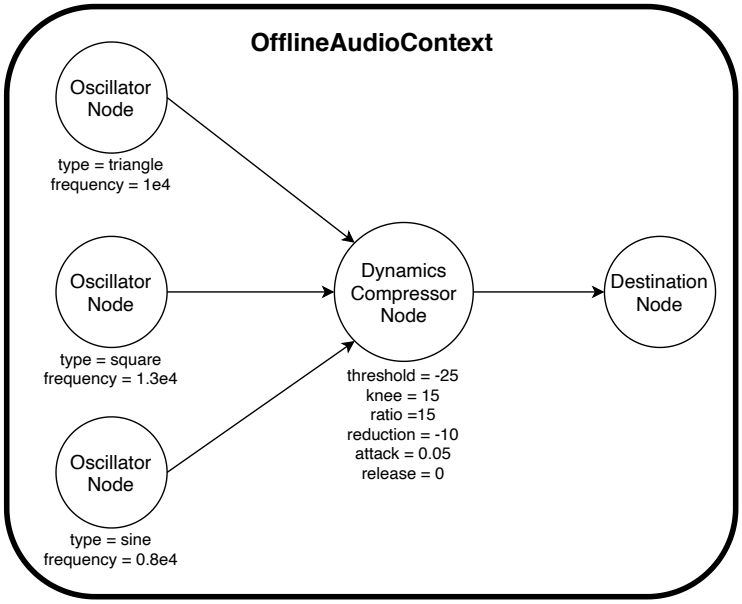

Fig. 21. Architecture of the network of `AudioNode` objects for the simple audio fingerprinting method.

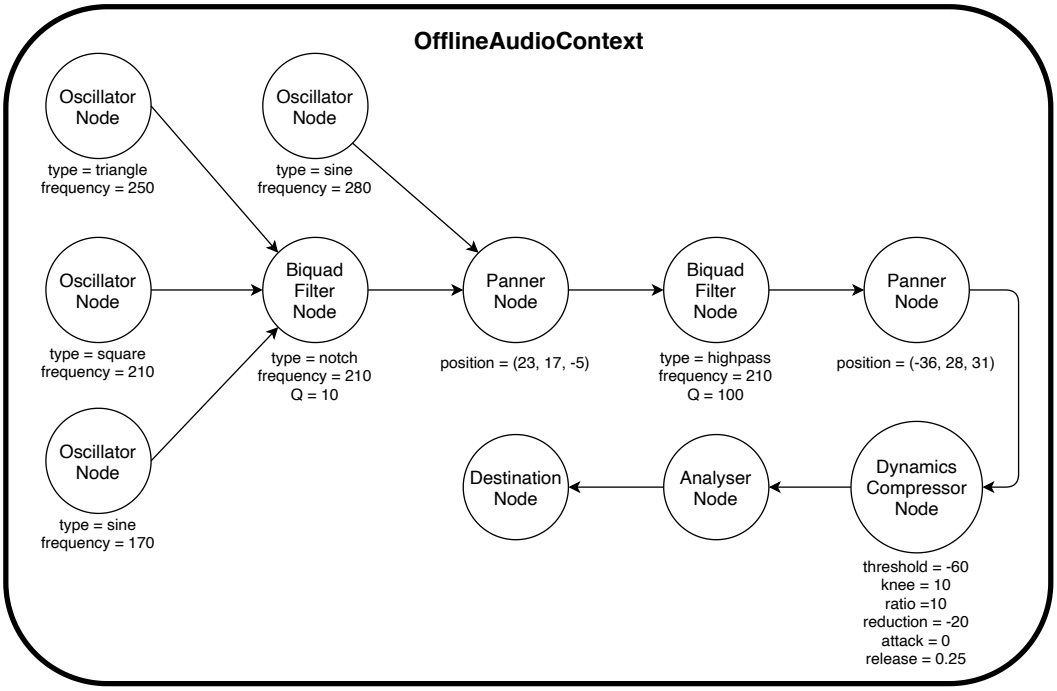

Fig. 22. Architecture of the network of `AudioNode` objects for the advanced audio fingerprinting method.

Table 9. The five attributes which collection time is missing in more than 1% of the overall, desktop, and mobile entries. The attributes are all asynchronous, and are ordered from the highest to the lowest proportion for the overall entries.

Attribute	Overall	Desktop	Mobile
WebRTC fingerprinting	24.04%	26.10%	20.31%
Audio FP advanced	13.20%	7.16%	31.57%
Audio FP advanced frequency data	13.20%	7.16%	31.57%
Audio FP simple	12.48%	6.39%	31.07%
List of speech synthesis voices	4.28%	1.15%	19.83%
Total entries	5,714,738	4,610,640	808,318

times, from which we extract 45 attributes. The single extracted attribute that is not concerned is the number of WebGL extensions. The remaining 189 base attributes and one extracted attribute are not impacted.

We measure the collection time of the fingerprints by the difference between two timestamps, one recorded at the starting of the script and the other one just before sending the fingerprint. Some fingerprints take a long time to collect that span from several hours to days. The high collection time of these fingerprints is explained below. Our fingerprinting script waits for the page body to load before collecting the attributes that require the body to be available, like the extension detection attributes. Due to a side effect, the script also waits for the body to be set before sending the fingerprint. We manage the waiting times using the `setTimeout` JavaScript function. The implementation of `setTimeout` in Chrome [43] and Firefox [73] throttles the time-out of the scripts that are running in inactive tabs to at least one second, and Firefox for Android increases up this threshold to 15 minutes. The waiting for the page body to load and the throttling applied by major browser vendors to inactive tabs result in the high collection time of some fingerprints. Less than 1% of the entries of each device group (i.e., overall, mobiles, and desktops) are impacted by these high collection time and considered as outliers.

We configured our fingerprinting script to collect the attributes in at most one second after the page body have been loaded. After this delay, the fingerprint is sent without waiting for all the attributes and the missing attributes do not have a collection time. All the attributes except five have less than 1% of the entries of each device group either having a high collection time (i.e., we deem the fingerprint an outlier) or missing the collection time. The five asynchronous attributes that miss collection time in more than 1% of the entries of each device group are presented in Table 9. The processing of these attributes take more time than the attributes which are a simple property. Moreover, they consist of complex processes that rely on components external to the browser (e.g., a real-time connection). As a result, the time limit can be reached before collecting the value and setting the collection time of these attributes. Mobile browsers show a higher proportion of missing collection time for the list of speech synthesis voices and the attributes related to the Web Audio API. This is due to the lack of support of these two APIs on mobile browsers at the time of the experiment [6, 7].

C KEYWORDS

In this section, we provide the keywords used to infer the browser family, the operating system, and whether the browser is a robot. We match these keywords on the `userAgent` JavaScript property

Table 10. The keywords and the exact UserAgent values that we use to detect robot browsers. The long values are cut at a blank space and displayed with indentations.

Blacklisted keyword	Blacklisted value
googlebot	mozilla/4.0 (compatible; msie 7.0; windows nt 6.1; trident/7.0; slcc2; .net clr 2.0.50727; .net clr 3.5.30729; .net clr 3.0.30729; media center pc 6.0; .net4.0c; .net4.0e)
evaliant	
bot.html	
voilabot	mozilla/5.0 (x11; linux x86_64) applewebkit/537.36 (khtml, like gecko) chrome/52.0.2743.116 safari/537.36
google web preview	
spider	mozilla/5.0 (windows nt 6.3; rv:36.0) gecko/20100101 firefox/36.0
bingpreview	mozilla/5.0 (macintosh; intel mac os x 10.10; rv:38.0) gecko/20100101 firefox/38.0

which is first set to lower case. We manually compiled the keywords by searching for meaningful keywords inside the collected UserAgents.

C.1 Robot keywords

We ignore the fingerprints that come from robots (i.e., an automated tool and not a genuine visitor). To detect these fingerprints, we verify that the UserAgent does not contain the keywords, nor is set to the exact values, that are listed in Table 10.

C.2 Device type

To infer the device type of a browser, we match keywords sequentially with the UserAgent of the browser. Table 11 lists the keywords that we leverage to infer each device type. The set of keywords can overlap between two device types (e.g., the UserAgent of tablet browsers often contain keywords of mobile browsers like `mobile`). Due to this overlapping problem, we apply the following methodology. The *mobile devices* are smartphones and do not include tablets. We check that their UserAgent contains a `mobile` keyword and no `tablet` nor `miscellaneous` keywords. To infer that a device is a *tablet*, we check that its UserAgent contains a `tablet` keyword and no `miscellaneous` keywords. The *miscellaneous devices* are game consoles and smart TVs. We check that their UserAgent contains a `miscellaneous` keyword. Finally, to infer that a device is a *desktop* computer, we check that its UserAgent does not contain any of the `mobile`, `tablet`, or `miscellaneous` keywords. In Table 11, we omit a `miscellaneous` keyword due to its size: "`opera/9.80 (linux i686; u; fr) presto/2.10.287 version/12.00 ; sc/ihd92 stb`".

C.3 Browser and operating system families

Table 12 lists the keywords that we use to infer the family of a browser, and Table 13 lists the keywords that we use to infer the operating system family of a browser. As a keyword can be in the UserAgent of two different families, we check the keywords sequentially in the order presented in the tables, and classify a device in the first family for which a keyword matches.

D ADVANCED VERIFICATION MECHANISM

Section 3.3.4 describes a simple verification mechanism that checks that the number of identical attributes between the presented and the stored fingerprint is below a threshold. In this section, we present the results obtained using an *advanced verification mechanism* that incorporates matching functions that authorize limited changes between the attribute values of the presented and the stored fingerprint. The methodology to obtain the datasets is the same as described in Section 4.3.3.

Table 11. The keywords that we use to infer the device type of browsers.

Mobile	Tablet	Miscellaneous
phone	ipad	wii
mobile	tablet	playstation
android	terra pad	smart-tv
iphone	tab	smarttv
blackberry		googletv
wpdesktop		opera tv
		appletv
		nintendo
		xbox

Table 12. The keywords that we use to infer the device type of browsers.

Browser Family	Keywords
Firefox	Firefox
Edge	Edge
Internet Explorer	MSIE, Trident/7.0
Samsung Internet	SamsungBrowser
Chrome	Chrome
Safari	Safari

Table 13. The keywords that we use to infer the operating system family of browsers.

Operating System Family	Keywords
Windows 10	windows nt 10.0
Windows 7	windows nt 6.1
Other Windows	windows nt, windows 7, windows 98, windows 95, windows ce
Mac OS	mac os x (but not ipad, nor iphone)
Linux-based	linux, cros, netbsd, freebsd, openbsd, fedora, ubuntu, mint
Android	android
Windows Phone	windows phone
iOS	ipad, iphone (but not mac os x)

D.1 Attributes matching

The *advanced verification mechanism* leverages matching functions for the comparison of the attributes. It counts the attributes that match between the two compared fingerprints, given the matching functions, and deems the evolution legitimate if this number is above a threshold. More formally, we seek to compare the stored fingerprint f to the presented fingerprint g. To do so, we compare the values $f[a]$ and $g[a]$ of the attribute a for the fingerprints f and g, using the matching function $\approx^a$. The *matching function* $\approx^a$ of the attribute a verifies that the distance between $f[a]$ and $g[a]$ is below a threshold θ^a. Finally, we deem g a legitimate evolution of f if the total number of matching attributes between f and g is above a threshold Θ.

Similarly to previous studies [33, 52, 101], we consider a distance measure that depends on the type of the attribute. We use the minimum edit distance [51] for the textual attributes, the Jaccard distance [106] for the set attributes, the absolute difference for the numerical attributes, and the identity function for the categorical attributes. The *distance thresholds* of each attribute is obtained by training a Support Vector Machines [45] model on the two classes of each month sample and extract the threshold from the resulting hyperplane. At the exception of the dynamic attributes that are required to be identical (i.e., the distance threshold is null) as they would contribute to a challenge-response mechanism [56, 80].

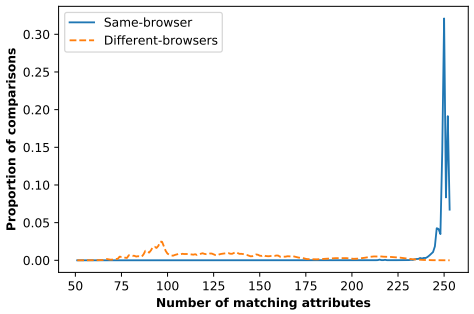

Fig. 23. The number of matching attributes between the same-browser comparisons and the different-browsers comparisons.

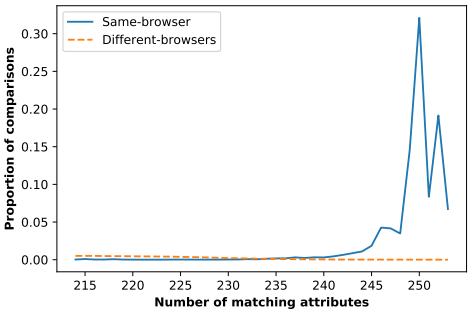

Fig. 24. The number of matching attributes between the same-browser comparisons and the different-browsers comparisons, starting from 214 matching attributes.

D.2 Distribution of matching attributes

Figure 23 displays the distribution of the matching attributes between the same-browser comparisons and the different-browsers comparisons, starting from 51 matching attributes as there are no observed value below. Figure 24 presents a focus that starts from 214 matching attributes, below which there are less than 0.001 of the same-browser comparisons. We can observe that the two sets of comparisons are well separated, as 99% of the same-browser comparisons have at least 235 matching attributes, and 99% of the different-browsers comparisons have fewer. The different-browsers comparisons have generally a fewer, and a more diverse, number of matching attributes compared to the same-browser comparisons. The different-browsers comparisons have between 51 and 253 matching attributes, with an average of 134.59 attributes and a standard deviation of 43.25 attributes. The same-browser comparisons have between 81 and 253 matching attributes, with an average of 249.45 attributes and a standard deviation of 3.69 attributes.

D.3 Distribution of match rates

Figure 25 displays the false match rate (FMR) which is the proportion of the same-browser comparisons that are classified as different-browsers comparisons, and the false non-match rate (FNMR) which is the inverse. The displayed results are the average for each number of matching attributes among the six month-samples. As there are no same-browser comparisons that have less than 235 matching attributes, the FNMR is null until this value. However, after exceeding this threshold, the FNMR increases as the same-browser comparisons begin to be classified as different-browsers comparisons. The equal error rate, which is the rate where both the FMR and the FNMR are equal, is of 0.66% and is achieved for 234 matching attributes.

D.4 Comparison with identical matching

The matching functions of the advanced verification mechanism leads to more matching attributes than identical attributes between two fingerprints. The higher number of matching attributes happens for the same-browser comparisons, but also for the different-browsers comparisons. This reduces the False Non-Match Rate (FNMR), but increases the False-Match Rate (FMR). Due to the FMR being higher, the equal error rate is slightly higher for the advanced verification mechanism that leverages matching functions.

Table 14 compares the results of the simple verification mechanism that leverages the identical attributes, and the advanced verification mechanism that leverages the matching attributes.

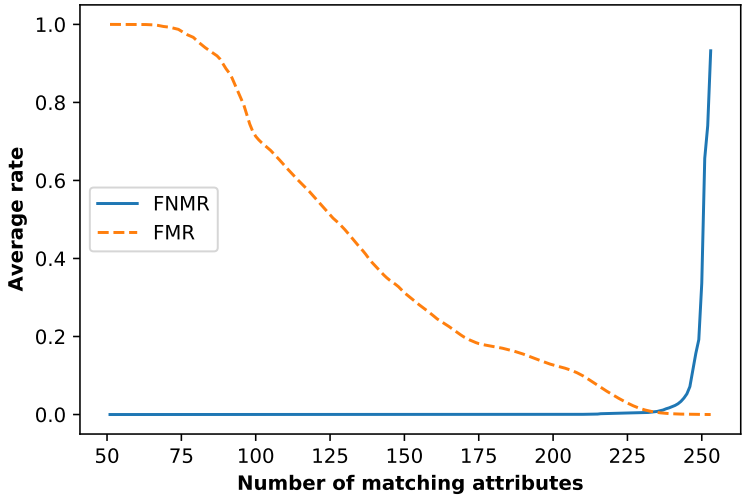

Fig. 25. False match rate (FMR) and false non-match rate (FNMR) given the required number of matching attributes, averaged among the month six samples.

Table 14. Comparison between the simple verification mechanism that uses identical attributes and the advanced verification mechanism that uses matching attributes.

Result	Simple	Advanced
Same-browser: range of identical or matching attributes	[72; 252]	[81; 253]
Same-browser: average identical or matching attributes	248.64	249.45
Same-browser: standard deviation	3.91	3.69
Different-browsers: range of identical or matching attributes	[34; 253]	[51; 253]
Different-browsers: average identical or matching attributes	127.41	134.59
Different-browsers: standard deviation	44.06	43.25
Equal error rate	0.62%	0.66%
Threshold of identical or matching attributes	232	234

Considering the matching functions only increases the average number of matching attributes for the same-browser comparisons by 0.81, whereas the increase is greater for the different-browsers comparisons at 7.18. The matching functions seem to contribute more to falsely linking different-browsers comparisons than same-browser comparisons. Due to this, the equal error rate is slightly higher for the advanced verification mechanism than for the simple one, respectively at 0.66% against 0.61%. Finally, we remark that the range of the identical attributes for the same-browser comparisons only goes up to 252 attributes. This is explained by our deduplication step during the preprocessing of the dataset (see Section 2.4.3) that removes the consecutive fingerprints that are identical. As a result, the same-browser comparisons have fewer than 253 identical attributes and are forcibly different. As for the different-browsers comparisons, it always goes up to 253 attributes as coincidence can make the fingerprints of different browsers match.

Table 15. The minimum, the average, the maximum, and the standard deviation (Std. dev.) of the distinct values, the normalized entropy, the minimum normalized conditional entropy, the median collection time in seconds, the median size in bytes, and the sameness rate of the attributes.

Property	Minimum	Average	Maximum	Std. dev.
Distinct values	1	7,633	671,254	51,298
Normalized entropy	0.000	0.090	0.420	0.094
Minimum normalized conditional entropy	0.000	0.008	0.181	0.021
Sameness rate	0.470	0.982	1.000	0.069
Median collection time (seconds)	0.000	0.125	2.179	0.425
Median size (bytes)	1	23.51	502	60.84

E ATTRIBUTES LIST AND PROPERTY

In this section, we provide the complete list of our 262 fingerprinting attributes and their property. Table 16 lists our attributes with their number of distinct values seen during the experiment (Values), their normalized entropy (N. Ent.), their sameness rate (% Same), their median size (Size), and their median collection time (Time). We refer to Section 5 for the distribution of the properties of the attributes: the number of distinct values, the normalized entropy, the minimum normalized conditional entropy, the sameness rate, the median collection time, and the median size. Table 15 provides the minimum, the average, the maximum, and the standard deviation of these properties among the attributes.

E.1 Attributes list

To stay concise, we replace the name of common JavaScript objects or of API calls by abbreviations. We denote D the JavaScript `document` object, M the `Math` object, N the `navigator` object, S the `screen` object, and W the `window` object. Additionally, we denote A an initialized `Audio Context`, AA an initialized `AudioAnalyser`, and AD the `A.destination` property. Finally, we denote WG an initialized `WebGL Context`, WM the `WG.MAX_` prefix, and WI the `WG.IMPLEMENTATION_` prefix.

Due to the diversity of JavaScript engines, some properties are accessible through different names (e.g., prefixed by `moz` for Firefox or `ms` for Internet Explorer). We denote `A.[B, C]` a property that is accessed either through `A.B` or `A.C`. If there is only one element inside the brackets, this element is optional. We denote `[...]` a part that is omitted but described in the corresponding attribute description.

Table 16. Browser fingerprinting attributes with their number of distinct values (Values), their normalized entropy (N. Ent.), their minimum normalized conditional entropy (MinNCE), their stability (% Same), their median size (Size), and their median collection time (Time). The comparisons of the attribute with itself and with the source attributes from which we derive the extracted ones are ignored for the MinNCE result.

Attribute	Values	N. Ent.	MinNCE	% Same	Size	Time
N.userAgent	38,863	0.394	0.046	0.978	115	0.000
Listing of N	1,660	0.207	0.009	0.989	502	0.001
Listing of screen	82	0.129	0.000	0.999	209	0.000
N.language	228	0.066	0.006	0.999	2	0.000
N.languages	1,448	0.094	0.010	0.998	17	0.000
N.userLanguage	124	0.036	0.001	1.000	1	0.000
N.systemLanguage	115	0.037	0.001	1.000	1	0.000
N.browserLanguage	52	0.036	0.000	1.000	1	0.000
N.platform	32	0.068	0.000	1.000	5	0.000
N.appName	5	0.003	0.000	1.000	8	0.000
N.appVersion	37,310	0.342	0.000	0.984	107	0.000
N.appMinorVersion	10	0.035	0.000	1.000	1	0.000
N.product	2	0.003	0.000	1.000	5	0.000
N.productSub	10	0.067	0.000	1.000	8	0.000
N.vendor	21	0.064	0.000	1.000	1	0.000
N.vendorSub	2	0.035	0.000	1.000	1	0.000
N.cookieEnabled	1	0.000	0.000	1.000	4	0.000
N.cpuClass	6	0.039	0.000	1.000	1	0.000
N.oscpu	60	0.071	0.000	1.000	1	0.000
N.hardwareConcurrency	28	0.086	0.025	0.999	1	0.000
N.buildID	1,351	0.076	0.010	0.989	1	0.000
[N.security, D.security[Policy]]	30	0.038	0.001	1.000	7	0.000
N.permissions	3	0.045	0.000	1.000	1	0.000
W.Notification.permission	5	0.043	0.001	0.999	7	0.000
W.Notification.maxActions	3	0.041	0.000	1.000	1	0.000
N.[msM, m]axTouchPoints	42	0.098	0.004	0.999	3	0.000
D.createEvent("TouchEvent") support	3	0.032	0.000	1.000	1	0.000
W.ontouchstart support	3	0.032	0.000	1.000	1	0.000
N.javaEnabled()	4	0.045	0.008	0.997	1	0.000
N.taintEnabled()	3	0.045	0.000	1.000	1	0.000
[[N, W].doNotTrack, N.msDoNotTrack]	11	0.085	0.012	1.000	6	0.000
N.connection support	3	0.022	0.000	1.000	1	0.000
N.connection.type	12	0.028	0.003	0.992	1	0.000
N.connection.downlink	91	0.032	0.003	0.995	1	0.000
N.[mozC, c]onnection.bandwidth	6	0.023	0.000	1.000	3	0.000
N.mediaDevices support	3	0.044	0.000	0.999	1	0.000
N.mediaDevices.getSupportedConstraints()	12	0.090	0.000	0.997	144	0.000
W.Intl.Collator().resolvedOptions()	311	0.097	0.000	0.999	115	0.005
W.Intl.DateTimeFormat().resolvedOptions()	1,849	0.154	0.011	0.996	111	0.003
W.Intl.NumberFormat().resolvedOptions()	260	0.070	0.000	0.999	138	0.001
W.Intl.v8BreakIterator().resolvedOptions()	75	0.046	0.000	0.999	1	0.000
N.getGamepads()	18	0.090	0.000	0.998	1	0.001

Attribute	Values	N. Ent.	MinNCE	% Same	Size	Time
W.InstallTrigger.enabled()	4	0.037	0.000	1.000	1	0.000
W.InstallTrigger.updateEnabled()	4	0.037	0.000	1.000	1	0.000
N.msManipulationViewsEnabled	5	0.052	0.000	1.000	3	0.000
N.[msP, p]ointerEnabled	9	0.051	0.000	1.000	3	0.000
D.msCapsLockWarningOff	3	0.039	0.000	1.000	1	0.000
D.msCSSOMElementFloatMetrics	4	0.039	0.000	1.000	1	0.000
N.[msW, w]ebdriver	6	0.048	0.001	1.000	3	0.000
W.Debug.debuggerEnabled	5	0.042	0.000	0.989	1	0.000
W.Debug.setNonUserCodeExceptions	4	0.042	0.000	0.989	1	0.000
new Date(2016, 1, 1).getTimezoneOffset()	60	0.008	0.001	0.999	2	0.000
Different Timezone at 01/01 and 06/01	3	0.005	0.002	0.999	1	0.000
S.width	1,280	0.192	0.005	0.987	4	0.000
S.height	1,016	0.188	0.015	0.987	3	0.000
W.screenX	3,071	0.125	0.047	0.925	1	0.000
W.screenY	1,181	0.126	0.049	0.925	1	0.000
S.availWidth	1,746	0.202	0.016	0.985	4	0.000
S.availHeight	1,353	0.268	0.058	0.984	3	0.000
S.availTop	460	0.060	0.003	1.000	1	0.000
S.availLeft	372	0.048	0.005	0.999	1	0.000
S.(pixelDepth, colorDepth)	14	0.031	0.001	1.000	5	0.000
S.deviceXDPI	249	0.073	0.000	0.993	1	0.000
S.deviceYDPI	249	0.073	0.000	0.993	1	0.000
S.systemXDPI	75	0.053	0.000	0.999	1	0.000
S.systemYDPI	75	0.053	0.000	0.999	1	0.000
S.logicalXDPI	6	0.039	0.000	1.000	1	0.000
S.logicalYDPI	6	0.039	0.000	1.000	1	0.000
W.innerWidth	2,572	0.263	0.023	0.957	4	0.000
W.innerHeight	2,297	0.388	0.181	0.906	3	0.000
W.outerWidth	2,481	0.293	0.074	0.909	4	0.000
W.outerHeight	4,046	0.327	0.117	0.872	3	0.000
W.devicePixelRatio	2,035	0.103	0.026	0.992	1	0.000
W.mozInnerScreenX	3,682	0.065	0.011	0.991	1	0.000
W.mozInnerScreenY	3,170	0.102	0.020	0.991	1	0.000
W.offscreenBuffering	4	0.067	0.000	1.000	4	0.000
S.orientation	3	0.044	0.000	1.000	1	0.000
S.[orientation.type, [moz, ms]Orientation]	26	0.107	0.003	0.994	21	0.000
S.orientation.angle	7	0.050	0.002	0.996	1	0.000
W.localStorage support	4	0.001	0.001	1.000	1	0.001
W.sessionStorage support	4	0.000	0.000	1.000	1	0.000
W.indexedDB support	3	0.002	0.000	1.000	1	0.000
W.openDatabase support	3	0.045	0.000	1.000	1	0.000
W.caches support	3	0.045	0.000	1.000	1	0.000
M.tan(-1e300)	15	0.087	0.002	1.000	19	0.000
M.tan(3.14159265359 * 0.3333 * 1e300)	12	0.085	0.000	0.999	18	0.000
M.acos(0.000000000000001)	4	0.058	0.000	1.000	18	0.000
M.acosh(1.000000000001)	7	0.071	0.000	1.000	24	0.000

Attribute	Values	N. Ent.	MinNCE	% Same	Size	Time
M.asinh(0.00001)	6	0.071	0.000	1.000	23	0.000
M.asinh(1e300)	6	0.058	0.000	1.000	17	0.000
M.atan(2)	3	0.018	0.000	1.000	18	0.000
M.atan2(0.01, 1000)	3	0.045	0.000	1.000	23	0.000
M.atanh(0.0001)	5	0.070	0.000	1.000	22	0.000
M.cosh(15)	8	0.058	0.000	1.000	18	0.000
M.exp(-1e2)	8	0.028	0.000	1.000	21	0.000
M.exp(1e2)	9	0.028	0.000	1.000	22	0.000
M.LOG2E	3	0.039	0.000	1.000	18	0.000
M.LOG10E	3	0.000	0.000	1.000	18	0.000
M.E	2	0.000	0.000	1.000	17	0.000
M.LN10	3	0.000	0.000	1.000	17	0.000
D.defaultCharset	71	0.075	0.001	0.999	1	0.000
Width and height of fallback font text	2,347	0.199	nan	0.998	11	0.099
W.[performance, console].jsHeapSizeLimit	24	0.083	0.002	0.991	3	0.000
W.menubar.visible	5	0.035	0.000	1.000	4	0.000
W.isSecureContext	4	0.045	0.000	0.999	5	0.000
S.fontSmoothingEnabled	4	0.042	0.001	1.000	1	0.000
new Date(0)	1,846	0.118	0.004	0.998	82	0.004
new Date("0001-1-1")	2,107	0.150	0.010	0.999	60	0.002
new Date(0) then setFullYear(0)	2,376	0.136	0.013	0.998	61	0.001
Detection of an adblocker	19	0.002	0.001	0.999	1	2.124
Firebug resource detection	3	0.037	0.000	1.000	1	0.054
YahooToolbar resource detection	3	0.037	0.000	1.000	1	0.056
EasyScreenshot resource detection	3	0.037	0.000	1.000	1	0.056
Ghostery resource detection	3	0.037	0.000	1.000	1	0.056
Kaspersky resource detection	3	0.037	0.000	1.000	1	0.057
VideoDownloadHelper resource detection	3	0.038	0.001	0.998	1	0.056
GTranslate resource detection	3	0.037	0.000	1.000	1	0.058
Privowny resource detection	2	0.037	0.000	1.000	1	0.057
Privowny page content change	3	0.000	0.000	1.000	3	2.162
UBlock page content change	4	0.000	0.000	1.000	1	2.151
Pinterest page content change	10	0.001	0.001	0.999	1	2.151
Grammarly page content change	3	0.000	0.000	1.000	1	2.152
Adguard page content change	3	0.000	0.000	1.000	1	2.179
Evernote page content change	3	0.000	0.000	1.000	1	2.156
TOTL page content change	3	0.000	0.000	1.000	1	2.153
IE Tab page content change	11	0.000	0.000	1.000	1	2.170
WebRTC fingerprinting	671,254	0.294	0.144	0.765	1	0.771
WG.SHADING_LANGUAGE_VERSION	23	0.103	0.000	0.996	18	0.001
WG.VERSION	247	0.123	0.008	0.995	10	0.000
WG.VENDOR	11	0.080	0.000	0.997	7	0.000
WG.RENDERER	14	0.089	0.000	0.996	12	0.000
WG.ALIASED_POINT_SIZE_RANGE	42	0.129	0.006	0.996	5	0.000
WG.ALIASED_LINE_WIDTH_RANGE	30	0.077	0.003	0.996	3	0.000
WM.VIEWPORT_DIMS	13	0.107	0.009	0.995	11	0.000

Attribute	Values	N. Ent.	MinNCE	% Same	Size	Time
WG.SUBPIXEL_BITS	9	0.039	0.001	0.997	1	0.000
WG.SAMPLE_BUFFERS	5	0.039	0.000	0.996	1	0.000
WG.SAMPLES	9	0.075	0.001	0.992	1	0.000
WG.COMPRESSED_TEXTURE_FORMATS	3	0.034	0.000	0.998	23	0.000
WM.VERTEX_UNIFORM_VECTORS	18	0.118	0.006	0.996	3	0.000
WM.COMBINED_TEXTURE_IMAGE_UNITS	19	0.079	0.003	0.996	2	0.000
WM.FRAGMENT_UNIFORM_VECTORS	18	0.109	0.004	0.996	3	0.000
WM.CUBE_MAP_TEXTURE_SIZE	11	0.084	0.008	0.995	5	0.000
WG.STENCIL_VALUE_MASK	8	0.050	0.000	0.996	10	0.000
WG.STENCIL_WRITEMASK	7	0.050	0.000	0.996	10	0.000
WG.STENCIL_BACK_VALUE_MASK	8	0.050	0.000	0.996	10	0.000
WG.STENCIL_BACK_WRITEMASK	7	0.050	0.000	0.996	10	0.000
WM.TEXTURE_SIZE	10	0.081	0.009	0.995	5	0.000
WG.DEPTH_BITS	7	0.047	0.000	0.996	2	0.000
WM.VARYING_VECTORS	19	0.121	0.009	0.996	2	0.000
WI.COLOR_READ_FORMAT	7	0.073	0.003	0.994	4	0.000
WM.RENDERBUFFER_SIZE	11	0.080	0.003	0.995	5	0.000
WG.STENCIL_BITS	5	0.016	0.000	0.997	1	0.000
WM.TEXTURE_IMAGE_UNITS	7	0.033	0.000	0.997	2	0.000
WM.VERTEX_ATTRIBS	8	0.017	0.000	0.997	2	0.000
WM.VERTEX_TEXTURE_IMAGE_UNITS	9	0.057	0.001	0.996	2	0.000
WI.COLOR_READ_TYPE	6	0.041	0.000	0.996	4	0.000
WM.TEXTURE_MAX_ANISOTROPY_EXT	9	0.029	0.000	0.997	2	0.001
WG.getContextAttributes()	54	0.114	0.009	0.995	138	0.000
WG.getSupportedExtensions()	535	0.209	0.027	0.990	401	0.008
WebGL vendor (unmasked)	27	0.115	0.000	0.995	9	0.000
WebGL renderer (unmasked)	3,786	0.268	0.073	0.991	20	0.000
WebGL precision format	25	0.071	0.001	0.996	114	0.001
Our designed WebGL canvas	1,158	0.263	0.023	0.990	64	0.041
Width and position of a created `div`	17,832	0.324	nan	0.940	18	0.084
Colors of layout components	7,707	0.153	nan	0.986	492	0.089
Size of bounding boxes of a created `div`	16,396	0.369	nan	0.470	31	0.197
Presence of fonts	17,960	0.305	0.110	0.996	198	0.450
Support of video codecs	84	0.114	0.001	0.999	78	0.002
Support of audio codecs	52	0.128	0.002	0.999	61	0.001
Support of streaming codecs	50	0.132	0.010	0.999	133	0.002
Support of recording codecs	7	0.069	0.000	0.999	140	0.001
List of speech synthesis voices	3,967	0.204	0.034	0.945	250	0.546
N.plugins	314,518	0.394	0.100	0.950	134	0.001
N.mimeTypes	174,876	0.311	0.017	0.982	112	0.000
A.state	5	0.082	0.000	0.999	7	0.000
A.sampleRate	16	0.070	0.019	0.997	5	0.000
AD.channelCount	5	0.036	0.000	1.000	1	0.000
AD.channelCountMode	4	0.036	0.000	1.000	8	0.000
AD.channelInterpretation	4	0.036	0.000	1.000	8	0.000
AD.maxChannelCount	20	0.058	0.003	1.000	1	0.000

Attribute	Values	N. Ent.	MinNCE	% Same	Size	Time
AD.numberOfInputs	3	0.035	0.000	1.000	1	0.000
AD.numberOfOutputs	3	0.035	0.000	1.000	1	0.000
AA.channelCount	5	0.067	0.000	1.000	1	0.001
AA.channelCountMode	5	0.037	0.000	1.000	3	0.000
AA.channelInterpretation	4	0.036	0.000	1.000	8	0.000
AA.numberOfInputs	4	0.036	0.000	1.000	1	0.000
AA.numberOfOutputs	4	0.036	0.000	1.000	1	0.000
AA.fftSize	3	0.035	0.000	1.000	4	0.000
AA.frequencyBinCount	3	0.035	0.000	1.000	4	0.000
AA.maxDecibels	3	0.035	0.000	1.000	3	0.000
AA.minDecibels	3	0.035	0.000	1.000	4	0.000
AA.smoothingTimeConstant	4	0.046	0.000	0.998	3	0.000
Audio FP simple	337	0.153	0.004	0.958	18	1.380
Audio FP advanced	561	0.147	0.001	0.953	17	1.644
Audio FP advanced frequency data	546	0.161	0.011	0.950	17	1.647
Our designed HTML5 canvas (PNG)	269,874	0.420	0.021	0.922	64	0.257
Our designed HTML5 canvas (JPEG)	205,005	0.399	0.001	0.936	64	0.262
HTML5 canvas inspired by AmIUnique (PNG)	8,948	0.353	0.002	0.986	64	0.031
HTML5 canvas inspired by AmIUnique (JPEG)	6,514	0.312	0.001	0.989	64	0.039
HTML5 canvas similar to Morellian (PNG)	41,845	0.385	0.034	0.947	64	0.037
Accept HTTP header	26	0.028	0.000	0.997	3	0.000
Accept-Encoding HTTP header	30	0.019	0.002	1.000	13	0.000
Accept-Language HTTP header	2,833	0.124	0.022	0.999	35	0.000
User-Agent HTTP header	20,961	0.350	0.002	0.978	108	0.000
Accept-Charset HTTP header	18	0.002	0.000	1.000	1	0.000
Cache-Control HTTP header	47	0.055	0.023	0.706	1	0.000
Connection HTTP header	2	0.000	0.000	1.000	5	0.000
TE HTTP header	2	0.000	0.000	1.000	1	0.000
Upgrade-Insecure-Requests HTTP header	2	0.000	0.000	1.000	1	0.000
X-WAP-Profile HTTP header	4	0.000	0.000	1.000	1	0.000
X-Requested-With HTTP header	151	0.004	0.000	1.000	1	0.000
X-ATT-DeviceId HTTP header	1	0.000	0.000	1.000	1	0.000
X-UIDH HTTP header	1	0.000	0.000	1.000	1	0.000
X-Network-Info HTTP header	4	0.000	0.000	1.000	1	0.000
Via HTTP header	4,272	0.007	0.003	0.999	1	0.000
Any remaining HTTP headers	5,394	0.095	0.042	0.899	192	0.000
Number of bounding boxes	15	0.062	0.008	0.998	1	0.197
Number of plugins	54	0.147	0.000	0.984	1	0.001
Number of WebGL extensions	28	0.176	0.000	0.991	2	0.008
Width and height of first bounding box	12,937	0.350	nan	0.486	30	0.197
Width and height of second bounding box	1,332	0.103	nan	0.941	1	0.197
Width and height of third bounding box	772	0.076	nan	0.965	1	0.197
List of widths of bounding boxes	6,690	0.299	nan	0.986	16	0.197
List of heights of bounding boxes	2,222	0.264	nan	0.474	14	0.197
Width of first bounding box	4,418	0.281	0.038	0.987	14	0.197
Height of first bounding box	1,848	0.246	0.070	0.490	14	0.197

Attribute	Values	N. Ent.	MinNCE	% Same	Size	Time
Width of second bounding box	471	0.085	0.002	0.998	1	0.197
Height of second bounding box	398	0.088	0.007	0.941	1	0.197
Width of third bounding box	224	0.060	0.004	0.999	1	0.197
Height of third bounding box	343	0.064	0.000	0.966	1	0.197
Width of a created `div`	15,473	0.316	0.007	0.940	6	0.084
Position of a created `div`	16,375	0.316	0.008	0.942	11	0.084
Width of fallback font text	1,029	0.184	0.024	0.998	5	0.099
Height of fallback font text	1,159	0.164	0.010	0.998	5	0.099
Color of ActiveBorder element	702	0.078	0.005	1.000	18	0.089
Color of ActiveCaption element	475	0.073	0.002	1.000	18	0.089
Color of AppWorkspace element	321	0.067	0.001	1.000	18	0.089
Color of Background element	2,917	0.074	0.017	1.000	16	0.089
Color of ButtonFace element	297	0.079	0.004	1.000	18	0.089
Color of ButtonHighlight element	264	0.058	0.000	1.000	18	0.089
Color of ButtonShadow element	343	0.076	0.001	1.000	18	0.089
Color of ButtonText element	104	0.004	0.000	1.000	12	0.089
Color of CaptionText element	123	0.014	0.000	1.000	12	0.089
Color of GrayText element	333	0.071	0.004	1.000	18	0.089
Color of Highlight element	1,088	0.097	0.016	0.987	17	0.089
Color of HighlightText element	89	0.049	0.001	1.000	18	0.089
Color of InactiveBorder element	334	0.060	0.000	1.000	18	0.089
Color of InactiveCaption element	441	0.062	0.001	1.000	18	0.089
Color of InactiveCaptionText element	265	0.088	0.006	0.999	15	0.089
Color of InfoBackground element	239	0.057	0.000	1.000	18	0.089
Color of InfoText element	96	0.003	0.000	1.000	12	0.089
Color of Menu element	376	0.087	0.004	1.000	18	0.089
Color of MenuText element	124	0.020	0.001	1.000	12	0.089
Color of Scrollbar element	275	0.072	0.000	1.000	18	0.089
Color of ThreeDDarkShadow element	75	0.071	0.001	1.000	18	0.089
Color of ThreeDFace element	297	0.062	0.000	1.000	18	0.089
Color of ThreeDHighlight element	261	0.048	0.000	1.000	18	0.089
Color of ThreeDLightShadow element	280	0.074	0.001	1.000	18	0.089
Color of ThreeDShadow element	339	0.075	0.000	1.000	18	0.089
Color of Window element	329	0.019	0.001	1.000	18	0.089
Color of WindowFrame element	140	0.069	0.000	1.000	18	0.089
Color of WindowText element	107	0.004	0.000	1.000	12	0.089