Big Data fingerprinting information analytics for sustainability

Anna Kobusińska, Kamil Pawluczuk, Jerzy Brzeziński

PII: S0167-739X(17)32996-5
DOI: https://doi.org/10.1016/j.future.2017.12.061
Reference: FUTURE 3899

To appear in: *Future Generation Computer Systems*

Received date : 25 July 2017
Revised date : 2 November 2017
Accepted date : 29 December 2017

Highlights

1. Available approaches to web tracking mechanism are analyzed in this paper, and big data fingerprinting information security for sustainability are extensively discussed.
2. Achievements and developments in big data fingerprinting discovery and exploration are presented.
3. To improve methods of fingerprinting collection, the fingerprinting analytics tool is proposed and developed.
4. The set of the most suitable fingerprints for sustainability are proposed based on the performed evaluation and analytics.

*

Big Data Fingerprinting Information Analytics for Sustainability

Anna Kobusińska[a,*], Kamil Pawluczuk[a], Jerzy Brzeziński[a]

[a] *Institute of Computing Science, Poznań University of Technology, Poznań, Poland*

Abstract

Web-based device fingerprinting is the process of collecting security information through the browser to perform stateless device identification. Fingerprints may then be used to identify and track computing devices in the web. There are various reasons why device-related information may be needed. Among the others, this technique could help to efficiently analyze security information for sustainability. In this paper we introduce a fingerprinting analytics tool that discovers the most appropriate device fingerprints and their corresponding optimal implementations. The fingerprints selected in the result of the performed analysis are used to enrich and improve an open-source fingerprinting analytics tool *Fingerprintjs2*, daily consumed by hundreds of websites. As a result, the paper provides a noticeable progress in analytics of dozens of values of device fingerprints, and enhances analysis of fingerprints security information.

Keywords: Big Data, Fingerprinting, Web Tracking, Security, Analytics

1. Introduction

In the recent years, Internet has become an essential part of everyday social and business life for billions of people around the world. Internet users exploit on a daily-basis a vast range of web-based applications, ranging from on-line shopping and banking to social networks. As more and more on-line business models are based on the necessity of distinguishing one web visitor from another,

*Corresponding author
Email address: Anna.Kobusinska@cs.put.poznan.pl (Anna Kobusińska)

various authentication approaches are applied [1, 2]. In addition to authenticate users and provide their secure access to web applications [3], also the ability to track them becomes essential.

The mechanism which has been so far heavily consumed for this purpose are *HTTP cookies* [4, 5, 6]. Once a web page is requested, a cookie containing a unique identifier is stored on the user's computer. Such a practice is fundamental for many websites to ensure a high level of usability. Yet, this mechanism has been recently under high public attention. Due to the continuous rise of privacy awareness in society, many people tend to either block or regularly remove cookies from their computers. Moreover, forthcoming laws and directives restrict the future usage of this storage type.

The past decade, however, showed that there are other mechanisms besides cookies that enable authentication and tracking web users. In [7], the authors proposed to combine computing device (e.g. desktop computer, smartphone, laptop or tablet) attributes in order to create, with a high likelihood, a unique device-specific identifier, called also a *fingerprint*. Fingerprinting is possible, because nowadays it is very unlikely that a set of random users, their devices, installed software or its settings will not differ in any way [8, 9, 10]. Information such as *User-Agent* header, screen resolution, hardware fingerprint (e.g. *audio*, *canvas*) or approximate location based on the IP address are just a few of the reasons for devices to differ. Despite above mentioned attributes are individually non-identified, once combined together, they hold an invaluable identification properties, which allow to uniquely discover the type of device and associate it with its user. The simplest solution to get the final user identifier (out of the attributes vector) is to apply a hash function to all of the information concatenated into one string. If none of fingerprint attributes changes over different visits of the user, such hash does not differ between consecutive executions of the algorithm, and therefore, it can be used to identify and re-identify a user visiting a web page.

Unfortunately, the device attributes that are used to generate fingerprints may often be altered, for example by a daily software updates or by modified personalization settings. Therefore, fingerprints need to be continuously verified and updated, in order to operate efficiently. This makes fingerprinting a complex process, demanding in terms of CPU power consumption, which takes into account large volumes of fingerprint attributes, having various characteristics and multiple data formats. Consequently, fingerprinting may be perceived as an approach of Big Data processing, which may be applicable as a complementary technique of Big Data analytics, providing a support especially in terms of web traffic and user behavior analysis, and in digital marketing opportunities. Among fingerprinting applications, various Big Data security-related analytics solutions can be mentioned. They include fraud detection, blocking abusive users, protection against account hijacking, as well as anti-bot and anti-scraping services, and many others. Fingerprinting analytics can also be adopted for multi-factor user authentication, which increases security, while optimizing user's convenience at the same time.

With the increasing popularity of fingerprinting, numerous fingerprinting studies started, analytics models were proposed, and fingerprinting analytics tools were developed [11]. Most of the proposed approaches focus on the evaluation of the new ideas, which could be turned into additional fingerprints. They also discuss the issues related to fingerprints diversity and stability, which are the primary challenges that each fingerprinting solution has to face. Although diversity and stability are the most important criteria for all of the fingerprint usages, yet many businesses are restricted with the additional conditions. The length of execution code, execution time and the length of the final fingerprint are crucial limitations of any real-time fingerprinting solutions.

Fingerprinting studies underline that the more appropriate data constitutes a device fingerprint, the more informative and accurate it is, allows to make better decisions, and provides better understanding of the sustainable user authentication. Thus, it is important to collect as many independent fingerprints as possible, so the samples are diverse enough to provide unique device recognition. On the other hand, to improve efficiency of web tracking and user authentication, the set of a large number of fingerprints supplied to an analytics tool should be optimized, for example by classifying and excluding unstable fingerprints from the process.

So far, despite a noticeable need of many companies that are trying to implement early solutions, above mentioned problems were not addressed by other theoretical studies. Thus, this study aims to implement various methods of fingerprint collection, and compare them accordingly to the most restrictive needs. In the paper, first the existing fingerprinting methods were discussed. A set of the most promising ones was chosen for evaluation, and was used by the proposed fingerprinting analytics tool. As a result of applied analytics, a set of most suitable fingerprints that efficiently and in a decent time provide sustainable user authentication was gathered. The proposed fingerprinting analytics was applied to the real fingerprinting data, received from thousands of different user browsers. The obtained data has been a subject of excessive analysis. As a result of cost-benefit evaluation, a set of features and respective optimal fingerprinting implementations have been chosen. The fingerprints selected in the result of the performed analysis may enrich and improve the existing fingerprinting analytics tools.

This paper is organized as follows. Section 2 describes the topic background: explains web tracking and available approaches, introduces the term of fingerprinting and its usages and challenges, as well as discusses the literature of the topic. Section 3 presents various features that can be fingerprinted and discusses their current status. The solution developed for the purpose of analysis is described in Section 4. Finally, Section 5 presents the obtained evaluation results and their discussion, while Section 6 brings final conclusions and proposes future work.

2. Device Fingerprinting — General Overview

2.1. Tracking Techniques

Web tracking is commonly known as assigning unique and possibly stable identifier to each user visiting a website. Its general purpose is to connect future web views of the same person or a computing device with historical ones. Most of all, it allows to serve personalized content and restore the visitors context.

The most common way of categorizing tracking is to divide it into storage-based and storageless techniques, depending whether those techniques use any of the storage mechanisms on the client side. A well known representative of storage-based technique are HTTP cookies [12]. According to Web Technology Survey statistics [13, 14], they are actively used on over 50% of websites globally. Half of them are persistent, meaning they remain on a visitors computer after closing the browser (until they expire or until deleted manually). Their rising popularity, brought up to the public the topics of privacy in the web and dramatically raised the awareness among people. Recent directives of the European Union, known as Cookie law [15, 16], require each website taking advantage of this mechanism to openly notify it. Thus, HTTP cookies are being increasingly deleted by privacy-conscious users. Additionally, some browser maintainers are starting to support this movement, e.g. Safari is blocking third-party cookies by default to protect unwary customers. All of that made cookies relatively unreliable. Fortunately, there are many alternatives.

High attention is recently directed towards *Web Storage* API, which was introduced in the newest HTML specification. It is already widely adopted by browsers (92% support[1]) and offers similar to cookies method of storing data, but for larger amounts. Usually, when the user requests a cookie removal, this storage is not cleared out, so the data still remains. Therefore, *Web Storage* is considered as modern cookies substitute for storing user identifiers more persistently.

ETags are identifiers set by a web server to specific versions of resources found under URLs [17]. Whenever a modification of the content occurs, a new tag is being assigned and sent together with the requested file. By exploiting this functionality aimed at cache validation, one can serve different *ETags* for each file request and thus, identify users. Browser cache could be used simi-

[1] `http://caniuse.com/#feat=namevalue-storage`

larly by serving files containing variable definitions of unique identifiers — they shall be read on the client side and attached to each further request. *Local Shared Objects*, known as *Flash* cookies, are another place to store data, same as Silverlights *Isolated Storage*, Internet Explorers *userData* storage or HTML5 *indexed database*. There are plenty of examples that could be exploited to serve as user identifiers storage, however most of them are having poor browser support or their reputation is infamous — knowing the history, reckless usage could end up with a law suit. A final solution for storage-based tracking is a JavaScript *Evercookie* [18, 19, 20]. This script produces extremely persistent cookies in the browser, using all possible methods at the same time. Whenever any of the identifiers from a particular source is removed, it is recreated using the remaining ones. A top-secret NSA document has been leaked by Edward Snowden in 2013, stating that *Evercookie* has been used to track users in *Tor* applications, i.e. browsers providing maximal privacy and anonymity. Obviously, using this script has all possible disadvantages reputation-wise.

On the other hand, storageless techniques do not employ any storage, and can be divided into three categories: history stealing, attribute-based and setting-based methods. History stealing is considered as attacks that are rather not visible across the web. CSS *history knocking* exploits the browser feature of marking visited links with different color (usually purple instead of blue). With JavaScript, one can write into HTML DOM some hyperlinks and test their CSS properties to determine whether the user has recently visited them. This attack has its origins in the past decade. Over time, browser maintainers were working to prevent exploiting similar features — some queries for computed hyper-link styles are being lied with false information about their appearance. Therefore, various timing attacks were invented to detect when browsers are trying to mislead. The battle between browsers and attackers is still in place today, in the name of users privacy.

On the other hand, attribute-based and setting-based methods are often referred to as *fingerprinting* (device, browser or user fingerprinting) [21], [22]. Fingerprinting focuses on collecting as many small pieces of information as possible. When those pieces are put together, they enable a reasonably unique device identification. Various categories of fingerprints could be determined, among which are low-level fingerprinting: hardware (CPU or GPU measuring) and network fingerprinting (comparing TCP/ICMP/AJAX clock skew); information-based fingerprinting: collecting available information, e.g *User-Agent*, JavaScript properties; behavioral/biometric fingerprinting: measuring mouse movement, typing, etc. On the other hand, fingerprinting could be divided into two categories according to the execution mode: passive (collection of already available data), and active (measuring, tracking or active querying in purpose of collecting additional information).

2.2. Fingerprinting Usages

While storage-based techniques are relatively easy to be noticed, fingerprinting is bringing the worst-class scenario for user privacy. It has the insidious property of not leaving any persistent evidence of device identification process that has occurred. Therefore, it has slightly wider applications. Some of the most important [23, 24, 25] are: identifying users on devices previously used for fraud, establishing a unique visitor count, advertising networks attempting to establish a unique click-through count, advertising networks attempting to profile users to increase ad relevance, profiling the behavior of unregistered users, linking the visits of users when they are both registered and unregistered and identify the user when visiting the site without authenticating.

If fingerprinting algorithm would be advanced enough to recognize most of the tested devices as unique, it could be used as a replacement for HTTP cookies. However, the reliability of the latter (if supported), will always outrank fingerprinting due its stability problems. Thus, combining strengths of both solutions is often the way to go. Inspired by *Evercookie*, fingerprinting could be used for re-spawning cookie identifiers. Instead of only storing them, servers could pair them with corresponding fingerprints. Whenever the cookie is lost, due to expiration or deletion, it could be regenerated based on stored fingerprint.

Additionally, previously impossible device recognition with IP addresses, because of many of them were hidden behind NAT [26, 27] to be addressed, together with fingerprinting may be quite successful. Since the set of translated devices is rather very limited, fingerprinting would be much more reliable if considered globally — the number of collisions would be much smaller. Obviously, IP address can serve as fingerprint itself, yet issue with its frequent change over time would have to be addressed [7].

2.3. Fingerprinting Obstacles

A primary obstacle the fingerprinting algorithm has to deal with is stability. Over time, the users browser or device is upgraded, which causes some fingerprints to change its value. Ideally, one should approach this problem by tracking the changes in certain ways. Once the browsers is updated, the *User-Agent* header is upgraded to a higher browser version string. Some of the installed add-ons are no longer supported and therefore temporarily or permanently disabled. This is one of the examples of fingerprints evolution. Such changes are mostly deterministic, so machine learning algorithms could make an effect in following them [28], [29]. Still, any abnormal user action, e.g. disabling cookies due to privacy awareness raised, installing a new font or change of device location, would bring unpredictable shift which is hard to deal with. Only if the adjustment is not serious, it is likely to be still detected.

Diversity is another obstacle fingerprinting has to cope with. All the information about particular device collected within fingerprinting, needs to be as unique as possible. There are many machines sharing the same configuration and having similar setting which fingerprint may be identical. Therefore, it is crucial to collect many and diversified fingerprints.

Measuring fingerprints diversity can be done with a mathematical tool — entropy. A distribution of a set of fingerprints is having 20 bits of entropy if randomly picked value is only shared with one among each 2^{20} devices. Entropy is defined as follows:

$$H(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_2 \mathrm{P}(x_i) \tag{1}$$

where $X = (x_1, x_2, ..., x_n)$ is a set of observed features, where $P(x_i)$ describes discrete probability distribution. If a website is regularly visited by a set X of different browsers with equal probability, the entropy is going to reach its maximum and could be estimated as $H(X) \approx log_2|X|$.

2.4. Related Work

In 2010, the Electronic Frontier Foundation (EFF) published a reference study [7] on browser fingerprinting. Relatively simple script has been developed and used to collect over 470,000 samples, among which 18 bits of entropy was observed. In total, 83.6% of unique users were recognized. According to the study, fingerprints were changing quite rapidly (chance for a change of at least one during primary 24 hours reached 37.4% while after 15 days raised to 80%), however it was relatively easy to track. Using basic string similarity algorithm, 99.1% of modifications were tracked (false-positives rate was 0.86%). Forged *User-Agent* header was not enough to mislead the detection.

For a couple of years, Princeton University, cooperating with Catholic University of Leuven, has been conducting relevant and valuable studies in the field of privacy on the web. Published in 2014 paper [30], presenting the problem of canvas fingerprinting, cookie re-spawning and syncing, brought serious media attention to these topics. Partially because of it, the score of 5.5% crawled sites exploiting canvas fingerprinting in 2014 dropped down to 1.6% in 2016. Cookie syncing analysis showed, that only around a quarter of third-party scripts is respecting users not willing to be tracked (who have used either opt-out cookies or set *Do Not Track* header). Created for the purpose of conducting privacy studies on large scale, *OpenWPM* web privacy measurement framework is regularly used for analysis of over a million top websites. According to recent results, tracking is especially popular among websites serving news. Scripts coming from particular companies that were present on over 10% of analyzed sites were only from the biggest players: Facebook, Google and Twitter. Nevertheless, browser add-ons such as *Ghostery* or *uBlock Origin* are dealing with those scripts quite effectively, except of very sophisticated and advanced ones that are hard to classify (same for fingerprinting only around 60-70% of scripts is blocked). Canvas fingerprinting of fonts were observed on 0.3% of websites while IP NAT address fingerprinting with *webRTC* API or audio fingerprinting were present only on about 0.06% of sites [31].

There are also plenty of websites aimed at raising awareness of tracking among Internet users. Many on-line fingerprinting tools [32, 33, 34, 35], exposing various browser features, have been developed — collected fingerprints are a subject of analysis for many similar studies. Moreover, some additional websites aimed at helping users to adjust their browsers protection are present [36], [37].

3. Classification of Fingerprint Categories

In Section 2.4, a short review of fingerprinting topic-related studies was presented. Within them, various fingerprinting tools and techniques have been already implemented and tested. Some of the fingerprinted features became popular for their confirmed stability and diversity, while some were classified as useless. This Section systematizes the available knowledge and discusses the possible improvements. It also presents an excessive list of fingerprinting features. Based on their current status, a set of features has been chosen for the further evaluation within this work.

Since the number of possible features to be fingerprinted is immense, this Section does not undertake to comment on all of them. Many of the features were omitted on purpose, while they may be considered outdated due to technology evolution, inapplicable for certain reasons, or simply, the author found them as bringing too little value for the study. Moreover, as the work is focused on browser fingerprinting, none of the behavioral (user) fingerprints, e.g. ones that measure/track user behaviors/characteristics, were included.

Fingerprints have been divided into two categories, based on the source of information: JavaScript code executed within the client browser or HTTP headers obtained on the server side. Additionally, JavaScript category has been divided into browser and device fingerprints, for the purpose of more accurate classification. The above mentioned HTTP-based fingerprints are all browser-specific, except of IP address which is a device property. However, the boundaries between this second-level categorization are often small enough to consider the feature as a member of both families.

3.1. Browser-Specific JavaScript Fingerprints

Ad-blocking add-on. In the era where advertisements are present on almost all websites, it was a matter of time for ad-blocking browser add-ons to be created. *AdBlock*, *AdBlock Plus*, *uBlock* and others, have been already widely adopted. Although there is no straight way to check for this add-ons to be installed, while browsing the web one can often notice banners telling to turn off any enabled ad-blocks in order to see the content — as the service's only income source is generated thanks to advertisements.

First solution, widely adopted across the web, tests for external script with specific name to be entirely blocked from loading by ad-block software. Yet, it could't be tested within this study since it assumes usage of only one core script that does not query for any additional resources. Second way is based on creating invisible `div` element, which `class` property contains one of the phrases: *ad*, *advertisement*, *adsbox*, *adsframe* and so on. Ad-blocking add-on filters are set to make such elements hidden, invisible or block them in other ways. Many tests to verify which method is the most efficient were developed. In the browsing private-mode of Chrome and Opera all of the add-ons are disabled by default. In Microsoft Edge they cannot be enabled even though they are installed. Therefore, by using ad-block detection, one must be aware that it may make the fingerprint unstable.

JavaScript binary-value properties. There are many properties exposed within JavaScript APIs (e.g. `window`, `navigator`) bringing valuable information. Most of the fingerprinting solutions available, are checking those values in *true-false* dimension only. However, it is not correct approach since different browser versions may handle them quite unexpectedly, for instance, returning *false*, *null* or *0* as the negative value. Treating them all as *false*, would be a rejection of precious data that is aimed to be collected. Moreover, another additional piece of information can be obtained by slightly more detailed querying — by adding vendor prefixes. Some properties used to be prefixed with *webkit*, *moz*, *ms* or *o* respectively for Chrome, Firefox, Internet Explorer and Opera browsers, prior the final standard was created. Thanks to them, developers were able to control inconsistencies between the browsers. Prefixes for certain properties are still working, even though they are often marked as deprecated. Such checks were included in the evaluation. Below some of JavaScript binary-value properties are presented:

- `AddBehaviour` function used to be a function available in some older versions of Internet Explorer. It was included in further analysis to verify if such fingerprint increases the overall entropy.
- Cookies — browsers are exposing cookie support by setting a property `navigator.cookieEnabled`. Seven different values of such a fingerprint can be observed among various browsers [38]. Additionally, actual ability to create a cookie was sampled since the first setting could be lied or overridden.
- DNT header — users are able to set *Do Not Track* flag, indicating whether they wish to not be tracked. Sadly, there is no public law to respect this setting. IE 10 was released with DNT header set to true by default — it brought a huge controversy. From that time, all of the browsers are not adding this flag unless the user explicitly wishes otherwise.
- Indexed database is another modern feature that could be deactivated or not supported. While testing it, there is a danger that the user will be prompted with permission dialog. That should only happen if the size of inserted value is relatively large.
- Web Storage mechanisms — local storage and session storage were tested similarly to cookies — both for the setting and actual ability to use the

mechanism. Additionally, any `SecurityError` that have occurred was treated as another negative setting.

- Open database — `window.openDatabase` method enables connectivity to web SQL databases. Yet, it is not a part of official HTML5 specification and hence, it is not well supported so the distribution of its values could be beneficial.

Navigator object. `appCodeName`, `appName` and `appVersion` functions are exposed through navigator object returning respectively code name, name and version information of the browser. Presumably all modern browsers `appCodeName` is Mozilla, for compatibility reasons, therefore it does not make much sense to collect this value. `appName`, for most of the browsers, should return Netscape, except of Opera and some versions of IE. `appVersion` exposes much more information but likely they are redundant with *User-Agent* header.

User-Agent. This header is commonly attached to HTTP requests that the browser sends to the server but its value can be also obtained from JavaScript level via `navigator` object. *User-Agent* contains a set of values that allow to identify the client application. In a reference study [7], this fingerprint entropy in isolation was equal to 10 bits, meaning it may be a source of valuable information.

Browser tempering. Some available fingerprinting solutions are attempting to detect whether user has been tempering with the browser[2]. However, creation of artificial fingerprints out of existing one is incorrect — there is no value coming from adding such fingerprints to the final solution. Collected fingerprint values which are faked, make the final fingerprint unique by definition. Adding additional flags will not increase the overall entropy but will negatively impact execution time.

[2] `https://github.com/Valve/fingerprintjs2/pull/44`

Flash technology fingerprints. Over last year, *Flash* world-wide usage in the web dropped by 2% to 5.8%. Installed *Flash* version could make another fingerprint, yet due to its low support, increase in entropy is rather small [7]. Additionally, some of the browsers do not support it in a private mode, similarly to add-ons. Therefore, the browser fingerprint based on *Flash* may change once the user enables incognito browsing [11]. However, having *Flash* enabled, it is really easy to yield complete installed fonts list, which unlike CSS or canvas solution outlined in the next subsection, can be simply accessed. Such fingerprint considered independently, is having high entropy, which could be even increased by taking into account the order of returned fonts [7].

Installed plugins and supported MIME types. The list of plug-ins with corresponding MIME types can be obtained with interfaces: `navigator.plugins` and `navigator.mimeTypes`, except of the Internet Explorer which uses *ActiveX* object and requires probing. To protect privacy, Firefox from version 29 also restricts the full access to these APIs and it is suggested to query for their exact names like in IE. Plugins are binary libraries built in the browsers and extending their functionality, e.g. PDF viewer engine. Most of them are bundled with browsers and therefore, may not increase the overall entropy.

3.2. Device-Specific JavaScript Fingerprints

Language. Exposed by `navigator` object `language` property (in some versions of the browsers `userLangugage`, `browserLanguage`, `systemLanguage` are also available), is supposed to return user preferred language, in a format described by RFC specification, e.g. *en-US*, *pl-PL* or *de-Latin-CH_1992*.

Platform. `navigator.platform` represents the platform on which the execution takes place. The set of possible values is not closed and the representation may differ from browser to browser[3]. Example values are: *Linux aarch64*, *MacIntel*, *iPhone*, *Nokia_Series_40* or *PlayStation 4*.

[3] `http://stackoverflow.com/a/19883965/1644291`

CPU class. This property is presumably present only in Firefox and Internet Explorer (under `oscpu` and `cpuClass` endpoint), while in Chrome it is a part of `appVersion`.

Timezone. Utilizing JavaScript `Date` object, one can request an *offset* which shall represent user system timezone setting within 15 minutes slots. Browsers may yield here quite unexpected numbers[4], which, properly interpreted, could make a valuable fingerprint.

Screen properties. `window.screen` object may be used to yield properties such as device screen color depth, resolution and available resolution. The latter is representing the space that may be consumed by system applications (without menu bars). In terms of fingerprinting resolutions, depending on which value is greater (width or height), the screen orientation is additionally determined. Again, by using it, some fingerprinting solutions are incorrectly creating another artificial fingerprint. On the other hand, orientation may be dangerous considering stability, as the users may change it quite often.

Pixel ratio. `window.devicePixelRatio` returns the ratio of the (vertical) size of one physical pixel on the current display device to the size of one CSS pixel. For some devices (systems), the value is known to be fixed so it may be strongly platform dependent.

Fonts. The complete list of fonts installed in the system can make another complex fingerprint. Browsers do not provide a way to retrieve it without usage of external plugins (Adobe Flash or Java), however there are hacks to obtain a partial collection.

All methods described below assume a brute-force testing for a specific list of fonts.

- Retrieving fonts with canvas — a canvas method `measureText` returns an object that contains width and height of a text, based on specified font

[4] `http://stackoverflow.com/questions/19618066/date-gettimezoneoffset-returns-a-non-integer-value`

family and size. If a font is not available on user's device, the browser will employ fallback font and return the same dimensions for both queries.

Internet resources suggest to use a considerable size of font (e.g. 70 pixels) and fallback to either *monospace*, *sans* or *sans-serif* as they represent fully-supported font families.

- CSS fonts querying — another font probing solution relies on the same idea as for canvas. It employs simple CSS properties comparison. Two span objects have to be appended to DOM while a bunch of measurement methods are applied to detect differences between both texts. It was suggested that usage of *m* and *w* characters for the test string would improve the result as they are the widest.

Touch support. It is a complex problem to reliably determine the touch support in the browser. Fortunately, reliability is not a primary concern for the best fingerprinting algorithms. Various browsers expose touch APIs through different endpoints so wide range of solutions were created: passive checks for methods availability, touch-related property tests or attempts to utilize the functionality.

Canvas fingerprint. Canvas is an HTML element used to draw basic 2D graphics on a web page. First fingerprinting studies used to draw on a hidden canvas two text strings with different colors, consisting from the perfect pangram: "Cwm fjordbank glyphs vext quiz" [30]. Such images translated to characters, using canvas `toDataURL` method, were found as very distinctive, stable and clearly dependent on graphics card used for rendering. With the time, more sophisticated ways for exploiting hardware specification with canvas were discovered. Latest solutions are making use of the following features (in different combinations):

- drawing a text with two fonts: using one of the fonts well supported among browsers and using the default *fallback font* (while providing a fake font name e.g. *fake-arial123-font*, the browser has to use a default one if the requested is not available)

- using text string as a perfect pangram, rarely with some digits or special characters
- using different colors
- verifying Unicode support by drawing a *smile* (e.g. smiling face with open mouth icon represented by character *U+1F603*)
- checking for canvas `globalCompositeOperation` support
- drawing two rectangles and checking if a specific point is in the path with `isPointInPath` method
- testing canvas winding support by drawing two objects on top of each other with `evenodd` setting
- testing for canvas blending support with three overlapping objects

Various combinations of tests discussed above were implemented to allow the emergence of best solution. Also, as the fingerprint is relatively big in size and taking long time to be computed, additional checks for classification of the features were included.

WebGL fingerprint. *WebGL* JavaScript API allows to draw on three dimensional canvas in the browser and used properly, makes another example of hardware fingerprinting. Images obtained with this technology can be translated into text the same way as for canvas fingerprinting, and therefore easily compared. Additionally, a variety of settings that may extend the fingerprint, can be accessed within `getParameter` and `getShaderPrecisionFormat` methods. In this work, one example of drawing algorithm was tested [11] and a wide set of environment locales were collected for comparison. Some of them could bring a prominent value e.g. in some older versions of Safari, it was possible to obtain the information which graphics card model is installed [39].

3.3. HTTP Protocol Fingerprints

Since the solution developed within this study is taking advantage of a back-end service, some additional server-side attributes could be collected. Each time a browser requests a resource from a server, a HTTP protocol exchange takes place. This Section discusses which request headers of such communication could make a valuable fingerprints.

User-Agent and DNT headers. *User-Agent* and DNT headers were collected both from JavaScript and HTTP protocol level in purpose of assessing whether these sources can be treated equally.

Accept- headers.* The examples of the headers that are not related to requested website but rather fixed to a resource type are: `Accept`, `Accept-Encoding`, `Accept-Charset` and `Accept-Language`. Considering a decisively active fingerprinting algorithm, one could sent many faked requests from user's browser for different kind of resources to increase fingerprint entropy. Yet, this study followed a passive approach due to strict efficiency limitations for created solution. Headers were collected only for the single request that is made in order to transfer the data to the server.

Headers order. Apparently, the order in which the request headers are served is not explicit and may vary among browsers. This way, an additional fingerprint could be obtained and analyzed [7].

IP address. IP address of a machine connecting to a web server is one of the basic properties necessary to establish a connection. If there had been enough addresses for everyone in the web, they would have made a complete source of identification. Yet, devices have to share their IP's or have them assigned dynamically. Even though these limitations exist, a decent piece of information is provided. One of the solutions to overcome these issues is to use a geolocation service and treat IP fingerprints as approximated locations.

3.4. Features Selected for Further Analysis

Due to the large number of features that can serve as fingerprints, we focus below only on the selected fingerprint features, which according to the authors seem to be the most promising during the user authentication (Table 1).

Table 1: Table of chosen attributes to be evaluated.

Feature	**Data source**	**Additional motivation**	**No**[5]
addBehaviour	JavaScript		1
ad-block add-on	JavaScript	finding best implementation; comparing classifiers	4
appCodeName	JavaScript		1
appName	JavaScript		1
appVersion	JavaScript		1
canvas	JavaScript	finding most efficient implementation	9
cookies	JavaScript	comparing passive vs active testing	2
CPU class	JavaScript	finding best implementation	2
DNT header	JavaScript	finding best implementation	2
fonts	JavaScript	comparing canvas-based with CSS-based implementations; finding most valuable subset of fonts	10
indexedDb	JavaScript		1
language	JavaScript	finding best implementation	4
localStorage	JavaScript	comparing passive vs active testing	2
openDb	JavaScript		1
screen pixel ratio	JavaScript		1
screen resolution	JavaScript		1
screen color depth	JavaScript		1
sessionStorage	JavaScript	comparing passive vs active testing	2
timezone	JavaScript		1
touch support	JavaScript	finding best implementation	6
User-Agent	JavaScript	testing usability of UA parser client library; comparing to `naviagator.app*` methods	9
webGL drawing	JavaScript		1
webGL properties	JavaScript	choosing meaningful properties	10
Accept	HTTP		1
Accept-Encoding	HTTP		1
Accept-Language	HTTP		1
DNT header	HTTP	comparing to JavaScript source	1
User-Agent	HTTP	comparing to JavaScript source	1

Most of the chosen features are obtained from the executed JavaScript code, which in practice has emerged as the client-side programming language of the web. However, we have also decided to analyze some of the most often used

[5]Number of different implementations — for evaluation of additional motivations, many supplementary implementations for specific fingerprinted features were created.

HTTP-based fingerprints. Since some fingerprint features (f.ex. *DNT* and *User-Agent* headers) can be obtained from both sources: browser-specific JavaScript and HTTP protocol, our additional motivation during the evaluation of those features was a comparison of the efficiency of fingerprints obtained from both sources.

It is important to note, that browser fingerprinting do not have any explicit law interpretations. Some of the fingerprints are having questionable reputation and thus, are denounced within specific societies. This study does not focus on the legal issues. Any possible usage of poor reputation-wise fingerprints was not intended. All the collected samples were gathered for educational purposes.

4. Evaluation Environment

In Section 2.4, some examples of existing long-term studies focused on fingerprinting methods evaluation were presented. They rely on dedicated websites created with the purpose of collecting fingerprints. The user visiting the study page may press a "fingerprint me" button in order to obtain his own fingerprint and equally, to support the work by sharing it. However, in a short range of time, such a solution would not generate enough samples to allow drawing reliable conclusions. Therefore, a different approach had to be followed.

The proposed fingerprinting analytics tool was preliminary described in [40]. It consists of three parts: fingerprinting script (called `bf.js`), back-end services and an analytics module. The aim of fingerprinting analytics is to distinguish and choose a set of fingerprints that will make possible stable user identification, and thus will provide sustainable authentication. To overcome the limitation of collecting fingerprints from a single dedicated web page, a script that can be attached to any website was created (which in fact is the target scenario of its usage). However, instead of the machine that serves particular domain to process the fingerprint, it shall be sent to another server that is responsible for data collection. Such solution implies many technological issues that had to be addressed. They are discussed in this Section altogether with a description of

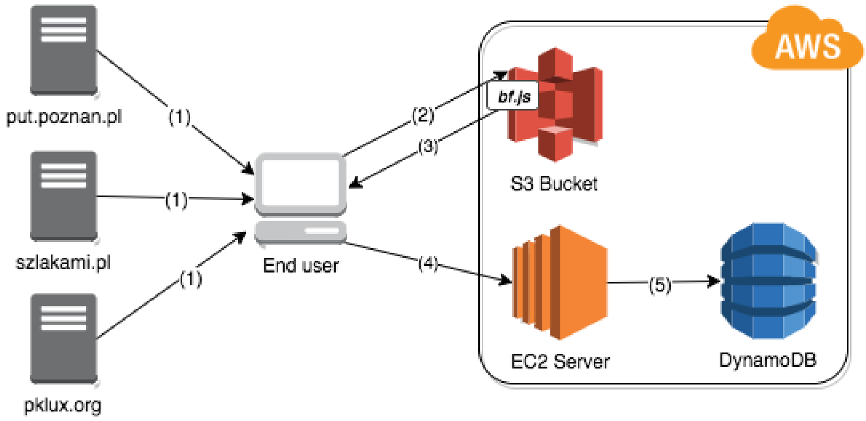

Figure 1: Fingerprinting process scheme

the setup.

General process of gathering fingerprint samples is presented in Figure 1. Amazon Web Services (AWS) are utilized as back-end services. The fingerprinting script `bf.js` is exposed within Amazon S3 Bucket and can be linked to any website. When a user enters one of the collaborating web pages, the script is downloaded and executed as one of the assets. The outcome is sent directly to the study server (Amazon EC2), where the analytics tool processes the data, appends backend-side fingerprints (HTTP request headers), and eventually stores them into DynamoDB database for further analysis. The statistics are generated with analytics module that fetches the data directly from Amazon.

4.1. Fingerprinting `bf.js` Script

The script, once triggered, collects all implemented fingerprints and sends them to the server, where they are stored in the database. However, instead of the database, various approaches that provide the reliability of obtained fingerprints can be used [41, 42, 43].

Communicating with the server. While creating fingerprinting framework, communication with the server was the first issue to be addressed. For security reasons, browsers restrict cross-origin HTTP requests initiated by scripts. Yet, there are certain exceptions that could be exploited. For example, a request for an image containing the data as GET parameter could be sent. Due to the character limitation of URL parameters [6], none of the solutions are applicable up to the size of 100 KB — the average size of a fingerprint obtained within `bf.js`. Therefore, *CORS*-enabled *AJAX* requests were used for transferring the data to the server. Within *CORS*, additional preflight HTTP request (by specification) is triggered before the actual request is made. This was a supplementary cost in performance that has to be kept in mind while evaluating the overall fingerprinting overhead.

Blocking redundant executions. As the script was going to be most likely linked on all of the sub-pages of the host website, each time the user would navigate or refresh the page, the fingerprinting process would be started. To prevent that, a cookie mechanism was implemented. Once the fingerprinting completed, it blocked its execution for next 3 minutes. Such suspension allowed to track long-term stability of fingerprints and at the same time, prevented flooding of the database with identical ones. This solution, as well as usage of *WebStorage* API during fingerprinting, brought the necessity to inform the users about usage of storage mechanisms, in accordance to European Union cookie law.

Ensuring ease of attachability to any web page. All of the modules were bundled together[7] and minified[8] to provide a single and minimal JavaScript file, that could be linked to any web page with ease. The overall size of the script was 70KB which could be considered a lot for a single asset[9], but was accepted by the cooperating websites. Fortunately, `bf.js` implemented many redundant

[6] HTTP protocol does not place any limit for the length of the URI. However the clients and the servers do only support the URLs up to a certain length. The rule of thumb is 2000 characters

[7] Bundling were achieved with *browserify* module (`http://browserify.org/`).

[8] *gulp* task runner (`http://gulpjs.com/`) with *gulp-uglify* package (`https://www.npmjs.com/package/gulp-uglify`) were employed for minification process.

[9] Similar solution *Fingerprintjs2* [11] script size is 33KB.

methods (for testing purposes), so the size of a production solution would be much smaller.

The only line that a website administrator had to add to the sources is presented on Listing 1. Note, it should be appended as the last line of the `body` tag to not affect any other scripts responsible for website behavior. Included `async` tag, which should take care of it by default, may be not respected by some of the browsers.

Listing 1: The code for linking `bf.js` to any website

```
<script type="text/javascript" src="https://
s3.eu-central-1.amazonaws.com/cdn.
alatar.eu/js/bf.min.js" async></script>
```

Ensuring efficient background execution. Even though the fingerprinting methods were written with a focus on minimal execution time, overall computation could take a noticeable moment, 2.5 second for a computer up to 4.5 second for a mobile device. It was mostly caused by canvas-related and font-related fingerprints, which are computationally expensive. Because JavaScript execution is made on a single timeline (except of `WebWorkers` which unfortunately could not be applied due to restriction of DOM manipulations), once the execution triggered, other scripts controlling the website could be starved of CPU time. Therefore, user experience on the website would be affected due to the delayed user interaction outcomes. This issue was solved by setting some timeouts with a break of 50ms between executions of consecutive methods. During those breaks, a JavaScript context switch could take place, if there were another scripts demanding to be resumed.

Ensuring lack of interaction with the users. Ideally, the script was supposed to be unnoticeable by the user in terms of performance, but regarding visual interactions, it should not bring any attention at all. The following precautions have been taken:

- All DOM manipulations (there were a few), were set to be hidden from user view by setting elements absolute position to inappropriately large negative value
- None of the API known for asking user for a permission (e.g. browser pop-ups geolocation API request for sharing device location) were accessed
- Names of all data stored locally were randomized to prevent overwriting existing values
- All the code was surrounded by a closure to protect the name-spaces of another scripts

Preventing exceptions. Since `bf.js` has taken advantage of a set of various JavaScript APIs that are often deprecated or not supported yet, there was a danger of how unexpected findings will affect the execution. Therefore, the whole script and all of the fingerprinting methods themselves, were carried out within `try-catch` statements. If an exception occurred, it would stay unnoticeable by the environment. In addition, the script would continue the fingerprinting of remaining features and attach the error messages to the results as a debugging information for improvement.

Since the script was written according to *ECMA Script 2015* specification, which is supported only by the newest versions of browsers, *babel*[10] compiler has been utilized to translate it into code understandable by older clients. Additionally, some JavaScript *polyfills* were appended to increase the number of supported browsers.

4.2. Back-End Services

Amazon Web Services were used as a back-end infrastructure for the whole solution. Their first and foremost goal was to provide high-availability and high-performance static files server for `bf.js`. As the number of study participants

[10] `https://babeljs.io/`

was unpredictable and any website could join the study at any time (by linking the script), the machine should be provisioned for high demand and easily scalable. Instead of creating virtual machine running Apache, Nginx or another type of server, Amazon dedicated solution for serving static files was utilized. S3 Bucket container is a space for files which is a part of Amazon content delivery infrastructure. It is used as assets server by Amazon itself, the same way it was used within this work.

Next element, constituted of EC2 service, provided endpoints for data collection. Created *t2.medium* virtual machine instance was running Amazon Linux RMI and Apache server. The latter served as a proxy to core functionality. It handled *AJAX* requests, initiated by `bf.js`, and through *WSGI* module executed its processing implemented in the Flask framework. Flask is a Python micro-framework suitable for applications exposing small functionality. Two endpoints were necessary to handle interactions, one for GET requests and one for POST. The first was a debugging routine which could be used to send exception message if such occurred on the client side. The second was gathering the fingerprints transferred as JSON payload of POST requests. It was also responsible for assigning unique cookie identifiers (for the purpose of tracking fingerprints stability), extracting and appending HTTP request headers to the dataset and finally, connecting to the database instance to dump the data. DynamoDB, an Amazon's distributed NoSQL solution, ensuring performance and high scalability, was used. Since the size of fingerprints (and therefore the requests) was substantial, it was provisioned with 15 MB per second throughput. In case it would not be enough for incoming traffic, it could be easily increased in a similar way the *t2.medium* instance could be upgraded. Fortunately, during the whole data collection period, there was no necessity to update any of the configuration.

4.3. Analytics Tools

All the results were generated with analysis tools written in Python. Amazon *boto3* library was utilized to access Dynamo database. The statistics functions

Table 2: The statistics of data collection from collaborating websites.

Website URL	Target country	Collection period	Collected samples	Unique users
`salsasiempre.pl`	Poland	33 days	10398	1743
`szlakami.pl`	Poland	28 days	3428	2506
`prononce.me`	France/Global	30 days	442	408
`carlosunlar.blogspot.com.br`	Brazil	20 days	312	177
`akai.org.pl`	Poland	35 days	290	136
`www.pklux.org`	Luxembourg	20 days	112	73
`edventurer.net`	Poland/Global	19 days	60	15
			15042	5038

and data cleansing were implemented using standard Python libraries and no additional tools were employed. Additional information about data processing was presented alongside the results in the next Section.

In order to collect a reasonable number of fingerprints, `bf.js` had to be linked to a minimal number of websites such that combined together visitors' traffic was analysis-considerable. A study page `e-fingerprint.me` had been created in order to find supporters. The data have been collected from 7 participating websites during approximate period of one month. In total 15042 records from 5038 users were obtained. The summary numbers are presented in Table 2. Luckily, the top two sources of samples presented opposed type of users activity — the first website allowed to collect fingerprints from returning users while the second one provided large amount of uniques.

5. Experimental Evaluation Results

The evaluation environment described in the previous Section allowed to obtain a reasonable number of samples for further analysis. In total, 15042 samples were collected (of the total size 1.36 GB). Each script execution queried the attributes listed in Table 1. This Section undertakes the task of their evaluation and analytics.

5.1. Evaluation Criteria and Data Representation

Evaluation criteria. Except of identification of the best possible fingerprinting implementations of certain features, each attribute has been analyzed according to the following criteria, which were proposed in [8][9]:

- Diversity — basic criterion for each fingerprinting study [44], a measure of how diverse is a set of samples calculated independently for each attribute as entropy. Additionally, a number of distinct and unique values in the dataset was counted.
- Stability — second cannon criterion stating how often a fingerprint is changing its value over the time [9][45]. Four characteristics were calculated for each method: total number of changes, average time distance between the changes, number of devices for which at least one alternation was observed, average percentage ratio of how many samples have been modified for these devices.
- Length of execution code — as the number of collected fingerprints increases, as well as libraries necessary for processing, size of the execution code becomes a limitation for some real-time-oriented businesses [46]. Thus, length of minified code for each method implemented in `bf.js` was included.
- Execution time — advanced fingerprints rely on time-consuming processing that makes another limitation [45]. Execution has been measured for each method independently so the average time could be calculated. 2% of the slowest records were dismissed since some edge values were enormously high, making the average overstated.
- Length of the fingerprint — in scenario when all the results are transferred to the server unchanged, their overall size is a shortcoming [45]. Average length of sent data was computed as the last criterion.

Data cleansing. Before the analysis, some essential data preprocessing was executed. Out of 15042 samples two processing sets were prepared:

- *data_unique* — a set of unique samples used as the base for all of the criteria evaluation, except of stability. It was created by filtering the samples by user cookie-based identifiers. For each user, only the earliest observed sample was taken. 8350 entries were removed so 6692 samples preserved. Yet, some cookies could have been removed in the meantime so their identical fingerprints could be stored under many *cookie-id*s. An important assumption has been taken — in such a small dataset with large number of fingerprinting methods, it is very unlikely that many collisions (two different devices having all of the fingerprints identical) could occurred. Hence, a subsequent filtering to remove identical fingerprints from the dataset was conducted. In total 1654 duplicates were dismissed, resulting in 5038 samples[11]. Considerable number of recognized duplicates confirms that cookies are being frequently removed by some users.
- *data_recurrent* — a set of 8146 samples constructed by filtering out all user entries from which only a single record was collected. In other words, the data for which stability over time could be evaluated was preserved in this dataset.

Summary table representation. The tests were summarized in the Table 3, which columns headers were abbreviated as follows:

- *dtC* — number of distinct values (diversity)
- *uqC* — number of unique values (diversity)
- *E(x)* — entropy (diversity)
- *modC* — number of changes (stability)

[11]For many tested attributes the overall amount of collected records differs due to various deployment dates.

Table 3: Summary table for conducted fingerprinting tests.

Test signature	dtC	uqC	E(x)	modC	modT	UmodC	UmodR	codeL	execT	fingL
accept	6	0	0.97	0				-	-	21B
acceptEncoding	10	1	0.62	4	22h 10m	3	20	-	-	13B
acceptLanguage	254	164	3.54	1	7d 21h	1	5	-	-	26B
adblock-adframe	6	0	1.08	54	23h 54m	22	44	300B	2ms	57B
adblock-adsbox	6	0	1.08	54	23h 54m	22	44	300B	3ms	57B
adblock-lib	5	0	1.04	49	1d 1h	22	43	500B	1ms	57B
adblock-lib-min	5	0	1.04	53	1d 1h	23	43	400B	1ms	57B
addBehavior	2	0	0.03	0				50B	1ms	1B
appCodeName	1	0	0	0				50B	1ms	7B
appName	3	0	0.05	0				50B	1ms	8B
appVersion	1056	842	6.34	108	7d 22h	95	30	50B	1ms	86B
canvas-advanced	864	450	8.08	90	4d 10h	74	34	900B	0.2s	21KB
canvas-basic	209	93	4.77	5	2d 20h	5	44	500B	11ms	8KB
canvas-fontArial	159	68	4.48	4	3d 13h	4	30	250B	2ms	580B
canvas-fontDigits	161	68	4.56	4	3d 13h	4	30	250B	2ms	677B
canvas-fontFake	166	78	4.03	4	3d 13h	4	30	250B	5ms	619B
canvas-fontSmiles	281	115	5.75	73	4d 21h	59	33	250B	4ms	573B
canvas-fontSpecialChars	156	70	3.98	4	3d 13h	4	30	250B	3ms	724B
canvas-moderate	646	312	7.38	76	5d 2h	61	29	500B	12ms	13KB
cookies	2	0	0.02	0				50B	1ms	1B
fontJs-sans-70px-821	2086	1688	9.07	187	6d 7h	166	29	1KB*	3.5s	5KB
screenColorDepth	4	0	0.74	0				50B	1ms	2B
screenDimensions	515	305	5.76	90	2d 22h	44	41	100B	1ms	14B
screenPixelRatio	6	0	0.82	3	14h 26m	3	4	50B	1ms	1B
timezone	22	7	0.74	1	4d 14h	1	3	50B	1ms	4B
touchSup-basic	2	0	0.76	6	2d 15h	2	23	150B	1ms	1B
touchSup-deprecated	2	0	0.75	6	2d 15h	2	23	50B	1ms	1B
touchSup-maxPoints	1	0	0	0				50B	1ms	1B
touchSup-modernizr1	2	0	0.75	6	2d 15h	2	23	50B	1ms	1B
touchSup-modernizr2	1	0	0	0				50B	1ms	1B
touchSup-msPointer	2	0	0.24	0				50B	1ms	1B
uaBrowserName	16	1	2.01	0				50B*	1ms	7B
uaBrowserVersion	236	112	4.47	120	7d 19h	109	31	50B*	1ms	9B
uaCpuArch	3	0	1.01	0				50B*	1ms	3B
uaDeviceType	4	1	0.87	0				50B*	1ms	2B
uaDeviceVendor	18	3	0.98	0				50B*	1ms	2B
uaEngineName	7	2	1.23	0				50B*	1ms	6B
uaEngineVersion	83	32	2.51	20	6d 12h	20	40	50B*	1ms	5B
uaOsName	13	3	1.32	0				50B*	1ms	7B
uaOsVersion	93	27	3.58	5	9d 7h	5	38	50B*	1ms	3B
userAgent-http	1105	833	7.46	124	7d 22h	113	31	-	-	104B
webGl	169	57	5.03	35	1d 16h	17	38	2KB	0.2s	4KB

- *modT* — average time between changes (stability)
- *UmodC* — number of users for which a change took place (stability)
- *UmodR* — percentage ratio of changes for these users (stability)
- *codeL* — code length
- *execT* — average execution time
- *fingL* — fingerprint record length

Data quality. Much of the fingerprinting research commented in early Sections of this work, relies on large databases of samples collected during long-term executions. This study was based on slightly over 15000 records that the author managed to collect. Moreover, the final evaluation was based on only 5000 cleansed entries. Yet, it is a recognizable amount of data allowing to draw reliable conclusions on certain aspects.

For the last three criteria (*codeL*, *execT* and *fingL*), which were the focus of the study, the number of collected samples was enough to make inferences. Concerning stability and diversity, one must be aware of some perspectives. Other studies fingerprinting algorithms may be concluded to provide e.g. 20 bits of entropy. This evaluation, having 5000 samples, could achieve at best $\log_2 5000 \approx 12.3$ bits. Obviously, both solutions cannot be compared due to different dataset sizes. Entropy measure should be considered in a relative sense, having the limit in mind. Secondly, the number of website's recurrent visitors is much smaller than the number of overall visits. Hence, the dataset used for stability measurements was even less referential than for diversity — stability results should be interpreted with a dose of uncertainty.

5.2. Presentation and Analysis of Evaluation Results

As expected, the highest entropy was observed for `appVersion` (6.34), `userAgent-http` (7.46), `canvas-advanced` (8.08), *webGl* (5.03 + properties) and `fontJs-sans-70px-821` (9.07) tests. Surprisingly, `screenDimensions` test was ranked with a decent 5.76

bits of entropy. `appCodeName` (0), `appName` (0.05), `cookies` (0.02) and some `touchSup*` tests (0) have proven to be useless in the overall evaluation. Such low entropies are caused by the fact that almost all collected samples values were identical.

The highest number of changes over time was observed for `fontJs-sans-70px-821` test (187) which inspected 821 fonts support. Apparently, its high entropy is correlated with stability problem. `adblock-*` tests, having a score of ∼50 changes and average time distance in-between equal to one day, uncovered unexpected instability. Again, `screenDimensions` result of 90 changes with a distance of 3 days was unforeseen. The length of code was the highest for canvas, *webGL* and font-related tests. However, the size of additional library for *User-Agent*, which was shared among `ua*` tests, was not included in the summary. It was a cost of additional 10KB, making these methods the most expensive in this category. The size of the list of fonts, used by probing tests, was also not included in the calculations since it was similarly shared.

Another issues that should be addressed are the size of canvas fingerprint records (up to 21KB) and execution time of `fontJs-sans-70px-821` test (3.5s).

Accept headers. HTTP header fingerprints proved to be a respectful and low-cost source of entropy. All three tests (`accept`, `acceptEncoding` and `acceptLanguage`) scored high results of 0.97, 0.62 and 3.54 bits. There were very few value changes during testing period so they also appear to be stable. Since HTTP headers were obtained on the server side, it is not necessary to deliberate on the last three criteria.

Ad-block add-on detection. Four methods of triggering ad-block add-ons were implemented and 9 attributes of detection were compared. The results are presented in Fig. 2. Usage of dedicated solution *BlockAdBlock.js* [47] was slightly less efficient than verification whether `div` with appropriate class tag (*adsbox* or *adframe*) was blocked. Testing `offsetTop` property of appended object turns out to be the best choice while `visibility`, `display` and `abp` gained the smallest score. Apparently, there is no need to combine many verification crite-

Figure 2: Results of ad-block fingerprinting

ria. Lastly, `adblock-*` tests instability was recognized but further investigation showed that all the changes originated from a small number of users (22). The author suspects they either have just installed the add-on or disabled it for some reasons.

`addBehaviour` test. collected almost identical values for all of the samples, resulting in only 0.03 bits of entropy. It is stable and cheap in execution but does not bring much of information.

`app*` properties, `ua*` parsing tests and User-Agent. The first question regarding *User-Agent* family of tests concerned any difference between the string obtain-

Table 4: Values obtained by parsing User-Agent header with UAParser.js library.

Device vendor		Browser		Engine		Type		Arch	
None	4689	Chrome	2746	WebKit	3827	None	4432	amd64	3186
Apple	657	Firefox	1607	Gecko	1607	mobile	1041	None	2629
Samsung	208	Mobile Safari	567	Trident	301	tablet	358	ia32	17
LG	61	IE	266	EdgeHTML	85	console	1		
Sony	49	Opera	187	Presto	10				
Lenovo	36	Safari	186	None	1				
Huawei	31	Edge	86	Webkit	1				
Nokia	29	Android	68						
HTC	24	Facebook	47						
HUAWEI	18	IEMobile	35						
ASUS	10	Opera Mini	9						
Motorola	10	Chromium	9						
Asus	3	Maxthon	7						
Xiaomi	2	WebKit	7						
Amazon	2	Silk	3						
BlackBerry	1	Vivaldi	2						
ALCATEL	1								
SonyEricsson	1								

able from JavaScript and HTTP request headers. In collected dataset such difference was observed in very few cases, therefore both data sources are redundant. JavaScript *User-Agent* was parsed with *UAParser.js* library to assess its usability in client-based classification. Since this fingerprint is relatively unstable, due to large amount of detailed information e.g. updated frequently browser version, it is worth to consider a separation of the data — to parse only the stable parts. Usage of the library was an additional cost of 10KB, which was not included in the summary table since it was shared by many tests. It provides a parser for the following features: browser name, version and engine; device type and vendor; OS name and version; CPU architecture. As expected, `uaBrowserVersion` tends to be updated regularly (120 changes, on average each 8 days) and at the same time scored the highest entropy (4.47). Remaining attributes were rather stable. Some of the collected values are presented in Table 4.

Last aspect to be addressed was determining any relationship or redundancy between `navigator.app*` fingerprints and *User-Agent*. `appCodeName` matched the expectations — values from all of the samples were Mozilla (0 bits of entropy). When it comes to `appName`, three different values were observed: Netscape (5806 entries), Microsoft Internet Explorer (16) and Opera (10), scoring only 0.05 points of entropy. `appVersion` results were much more diverse (6.34) and the list of unique samples was large. Many values (3960) concatenated with Mozilla foreword proved to be identical with *User-Agent* but remaining 1872 varied. 1514 samples were equal to *5.0 (Windows)*, the rest represented mostly subsets of the User-Agent strings but rarely added extra information.

Summing up, for most of the devices `navigator.appVersion` is redundant with *User-Agent* but in some cases it brings additional diversity. Yet, due to its high instability, similar to raw *User-Agent* header, for most of the usages it is rather impractical.

Canvas fingerprinting. Since this fingerprint was very popular within past years, many different ways of implementation, described in Section 3.2, were discovered. 12 canvas fingerprint tests were collected to answer the question which properties are the most valuable. Table 5 presents the characteristics of each test. Conclusions are following: the canvas size (width and height) is having considerable impact on the entropy. While all the drawn elements are bigger, number of unique fingerprints is significantly larger and the entropy increases. Moreover, tests for blending and winding support improved the overall result. The surprisingly high score of entropy was achieved by the smile icon rendering test. Also, unexpectedly, it turned out that the usage of fake (fallback) font has lower entropy than the usage of widely-accessible *Arial* font, even though it registered a larger number of unique and distinct values. Finally, adding a number to a text increased overall diversity.

The most advanced canvas test (`canvas-advanced`) obtained 8.08 bits of entropy. It is a significant score, however other criteria must be considered. Apparently, it is quite unstable (90 changes each 4.5 days), time consuming (0.2s) and its length is the highest from all collected fingerprints (21KB). Individual tests imply that the *smile* icon (`canvas-fontSmiles`) is the primary

Table 5: Components of canvas fingerprinting tests.

Test signature	Width	Height	Text complexity	Icon	Blending	Winding
canvas-basic	350	40	high			
canvas-advanced	550	200	high	X	X	X
canvas-advanced-min	200	60	high	X	X	X
canvas-moderate	550	200	high	X		
canvas-moderate-min	200	60	high	X		
canvas-blend-winding	550	200			X	X
canvas-blend-winding-min	70	65			X	X
canvas-fontSmiles	30	25		X		
canvas-fontFake	60	20	fake font			
canvas-fontArial	60	20	arial font			
canvas-fontDigits	60	20	only numbers			
canvas-fontSpecialChars	60	20	only special chars			

source of instability and, at the same time, of entropy. The bigger the canvas and drawn elements are, the higher the entropy, instability and the execution time. The only stable element seems to be the font drawing (`canvas-basic`, `canvas-font*`). Notwithstanding, the average fingerprint size of 21KB is too large for most. Luckily, the usage of a hash function can solve this issue if additional uniqueness deterioration is acceptable.

Cookies and Web Storage API support. Cookies, local and session storage were tested both using JavaScript properties (e.g. `navigator.cookieEnabled` indicating the setting) and with *active* evaluation with the following scenario: get storage handle, write some data into it, probe it for saved data existence, remove the data. If the check for saved content failed or an exception was raised, storage mechanism could be considered as disabled. The results (Table 6) reveal that such method was successful in detecting a few "lied" situations for local and session storage, while for cookies, property value was always providing the same answer. Unfortunately, even though storage fingerprints are stable and execution low-cost, their small entropy make them relatively irrelevant. It it also worth noticing, that only 2 distinct values were observed for cookies test while larger studies collected up to 7 configurations (Section 3.1). It confirms that small amount of collected data does not allow to draw widely applicable

Table 6: Results of storage support fingerprinting.

Test name	Negative	Positive	Exception
localStorage	9	5818	5
localStorage-active		5736	96
sessionStorage	9	5820	3
sessionStorage-active		5740	92
cookies	10	5822	
cookies-active	10	5822	

conclusions.

CPU class. In 95% of cases `navigator.cpuClass` did not return any value. 259 devices returned *x86*, 40 yielded *ARM* and *x64* was observed twice, all resulting in 0.25 bits of information. `oscpu` property returned much more interesting results, the ratio of empty values was 72%. Unexpectedly, it does not only concern CPU architecture but also OS version, making the entropy higher (1.76). Since both fingerprints were stable and their execution cost was negligible, such consideration in independence makes them a good choice for any algorithm.

Do Not Track *header.* This fingerprint was collected in JavaScript using two different objects, `navigator` and `window`. The obtained results were exclusive and they did not cover with the back-end side values. Table 7 presents observed configurations and their count. The fact that it is not clear what is the real user setting does not prevent these attributes from being useful in the fingerprinting process, thanks to relatively high entropies in comparison to small numbers of distinct values (2 or 3). Paradoxically, a feature that was created to protect privacy proved to be a valuable addition for this study.

Fonts fingerprinting. Among two methods of fingerprinting fonts, canvas and CSS, the more efficient one was intended to be uncovered. In a very early stage of the samples collection, it was already clear that CSS-based method is much more attractive than canvas probing. Because canvas tests were affecting overall processing time substantially, they were entirely removed from `bf.js` and not included in the summary table.

Table 7: *Do Not Track* values distribution.

	`navigator.DNT`	`window.DNT`	DNT (HTTP)	Count
	F	F	null	3765
	T	F	null	1390
	ms	F	null	7
	F	T	null	5
	T	F	1	411
	F	F	1	29
	ms	F	1	9
	F	T	1	215
total(T)	1818	220	664	*

Firstly, the average execution time of canvas-based font probing was roughly three times slower. Moreover, CSS detection slightly outranks canvas but in both methods efficiency is almost complete (assessed with manual verification), and CSS probing for foreign fonts containing exceptional characters (e.g. Japanese alphabet), even though there were not included in the test string, detected the font while canvas method did not. The authors suspect that CSS methods reserve the space (maximal height) for any character supported by a font, even if not printed explicitly. Also, in some browsers discrepancies of 1 pixel were observed. Therefore, the tests were improved to meet this margin of error. Additionally, the results revealed that usage of a test string containing full alphabet or the one chosen for fonts entropy assessment (*adfgjlmrsuvwwwwz7901*) increased the detection rate in comparison to the string proposed in other studies (based on *m* and *w* letters). Also, test string size of 70 pixels produced almost identical results as 180 or 200 pixels, and *monospace* font was slightly more effective than *sans-serif*, both for CSS and canvas tests. The only drawback of CSS method is the fact that it requires to be executed in the user's DOM, which brings a danger of influencing website appearance (canvas works in the background).

There were two additional observations which remain unsolved. Firstly, for unknown reasons, drawing with *monospace* as fallback font was on average 10 times faster than drawing using *sans-serif*. The authors did not find any confirmed explanation for this fact. It is suspected that *monospace* tests could have been optimized after *sans-serif* checks were run, although no particular execution order was assured. Secondly, drawing strings of size 200 pixels were twice faster than 70 pixels in CSS-based tests. The same possible explanation applies.

Another important aspect of fonts evaluation is determining a subset to be used for probing. A font that is not supported for each user nor is present in all the samples, will not allow to distinguish devices. Maximum entropy (1 bit) is reached when a font is present in exactly half of the data. Yet, choosing only such fonts will not maximize the output since many sets are strongly dependent. Therefore, an excessive list of 821 fonts was prepared and for all of them, a sample was collected. An iterative entropy maximization algorithm was executed in order to find optimal collection. Fig. 3 presents how big sets were necessary to obtain particular values of entropy. To achieve 6 bits result, in the best scenario the following 9 fonts were used (ordered from the most valuable): *Open Sans*, *Brush Script MT*, *Estrangelo Edessa*, *Gadugi*, *Roman*, *Papyrus*, *MT Extra*, *Wingdings*, *Segoe UI Semibold*. Above 8 bits, the number of fonts required to improve the entropy increases drastically. After reaching 9 bits the remaining 746 elements almost did not improved the result. It shows how important choosing the right collection is. It is essential not only for the diversity but also for the code execution time (3.5s) and stability (187 changes, 6 days), as this fingerprint achieved the worst results in both categories. Reducing the set of fonts from 821 to 100 would decrease the average time necessary for probing to around 0.4s which may be acceptable in certain usages. Stability metrics should improve as well, although `fontJs-sans-70px-65` test probing for only 65 fonts still presents alarmingly high instability (132 changes each 7 days). A short investigation revealed three main categories of changes that have occurred: (1) single font installation; (2) a large set of fonts changing the status from absent to present; (3) single font fluctuations.

The first two categories may denote that the user has installed an additional font or a new software. Unfortunately, there is nothing that can be done to

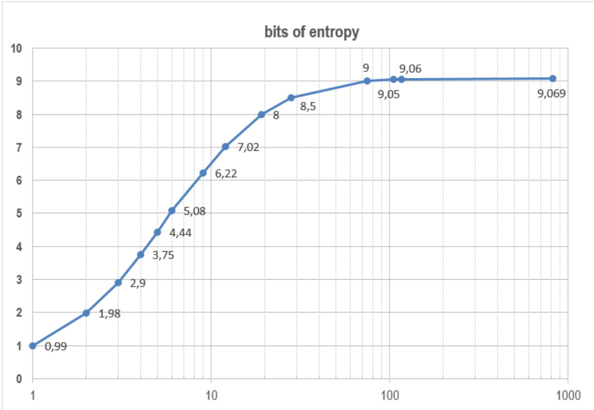

Figure 3: Number of fonts necessary to obtain particular level of entropy.

prevent them. Yet, often status changes of a particular font are quite unlikely to be caused by a user action. Thus, the latter category suggests either a field for detection algorithm improvement or necessity to investigate the cause in a deeper manner.

`indexedDb` and `openDatabase` fingerprints. 7 different values were observed for indexed database support, providing 1.31 bits of information. Open database fingerprinting scored 0.94 entropy with true-false setting. Both features were stable and easy to obtain so are worth to be included in the production solutions.

Language setting. 4 methods of obtaining language were implemented. Broadly supported (99.9%) `navigator.language` property presented 2.1 bits of information. Remaining tests returned a result in only 5% of cases and as their values were mostly equal, they barely achieved any entropy. Yet, thanks to a decent stability and low cost execution all of the features are worth consideration.

Platform fingerprint. has changed its value only once, so it is one of the most stable. 16 distinct values with 3 uniques were found in the dataset (1.57 entropy).

Screen properties. Among both `screenColorDepth` and `screenPixelRatio` tests, stable but rather similar values were collected, providing 0.74 and 0.82 bits

Table 8: Screen resolution instability investigation.

Test signature	dtC	uqC	E(x)	modC	modT	UmodC	UmodR
screenAvail	501	260	**5.71**	**90**	2d 22h	44	41
screen	153	64	4.06	65	2d 3h	29	36
screenDimensions	515	305	**5.76**	**90**	2d 22h	44	41

of entropy. However, screen dimensions method yielded surprisingly diverse (5.76 bits) and unstable results (90 changes, on average every 3 days). Instability was not expected since the test did not take into account the screen orientation. It was analyzed what entropy loss it implied — it was only 0.25 bits. Additional tests were run to investigate the source of instability (Table 8). Both methods frequently yielded different values for the same users, although `window.screen.availHeight` and `availWidth` prevailed the final result. Some changes were marginal (e.g. 404 pixels to 401 pixels) and their cause should be further investigated. Yet, many changes appear to be a switch to entirely new resolution of the same device or to an external display (rarely since color depth and pixel ratio did not change).

Timezone. results with 22 distinct and 7 unique values scored only 0.74 bits of entropy. Yet, this fingerprint is also very stable and execution low-cost so worth a consideration.

Touch support detection. The evaluation of 6 detection methods (Table 9) suggests, that the three could be used redundantly as they are all marked by the same devices as touch-enabled (25% of the dataset, 0.75 entropy). `touchSup-maxPoints` test and the second part of *Modernizr* library check method returned *false* for all of the devices. As Internet Explorer property `msPointer` marked additional devices as supported (0.24 bits), an ideal solution could make use of a combination of these features.

WebGL fingerprints. Besides collecting *WebGL* drawing fingerprint, 10 categories of properties were collected. Their high entropy makes them valuable, yet many samples have changed over the time (on average after 36 hours). As

Table 9: Results of touch support detection tests.

touchSup-basic	F	**T**	T	T	F	F
touchSup-deprecated	F	**T**	T	F	F	F
touchSup-modernizr1	T	**T**	T	F	F	F
touchSup-msPointer	F	F	**T**	F	**T**	F
touchSup-maxPoints	F	F	F	F	F	F
touchSup-modernizr2	F	F	F	F	F	F
configuration count	1	**1060**	10	3	182	3680

Table 10: *WebGL* fingerprints dependence analysis.

Test signature	dtC	uqC	E(x)	modC	modT	codeL	execT	fingL
webGl	169	57	5.03	35	1d 16h	2KB	0.2s	4KB
webGlProp-*	366	171	5.53	57	23h 46m	5KB	0.1s	682B
webGl*	459	225	6.31	73	1d 3h	6KB	0.4s	5KB

most of the tests manifested a similar performance, they do not allow to draw any conclusions independently. Additional evaluation was executed to asses the attributes together (Table 10). By combining drawing fingerprint with all properties, only 6.31 bits of entropy were achieved. In total 73 values have changed within a relatively short period of time, namely 27 hours. As for the cost of 0.4 seconds of execution time, the great length of code (6KB) and the final sample size of 5KB, this study does not allow to conclude that *WebGL* features are a necessary addition to any fingerprinting algorithm.

5.3. Evaluation Summary

In the paper, fingerprinting analytics was applied to study various fingerprinting approaches, uncover the existing correlations among fingerprints and choose the most appropriate ones for sustainable user authentication.

The proposed analytics tool was used to independently analyze and discuss the obtained results of performed tests. Based on those results, in this Section a selection of the most efficient features that could make the client-side production fingerprinting algorithm is conducted. Additionally, some important observations useful in creating more advanced solution that utilizes a server-side logic (and HTTP-based fingerprints) are summarized.

Client-side solution. Weighting the expectations from an optimal fingerprinting script, the following key points were summed up to serve as the criteria of the final selection:

- The script should not fingerprint any of the features classified as unstable.
- As many features as possible should be employed to ensure maximal diversity. Even if the fingerprint independent entropy is barely recognizable, but all the other criteria are matched, such feature should be included in the algorithm (the number of samples collected within this study is not significant enough to come up with a conclusion of permanent attribute rejection).
- Execution time of the script should not exceed 0.5s on average — many of the usages are aimed on blocking abusive users which should be executed as soon as they enter a website.
- A size of the final code bundle should be minimized to reduce the download time and save the bandwidth on mobile devices.

The features that meet the requirements are presented in Table 11. A few of the implemented tests have been concluded to need an improvement in order to match the criteria. Thus, with the purpose of measuring the characteristics of the algorithm created from an optimal set of implementations, the dataset was translated into a form of a results yielded by improved fingerprinting methods. The only issue was a lack of the real world execution time data — an estimation had been made based on the old methods performances.

The result achieved by all fingerprinting methods together, implemented in `bf.js`, were compared with the fingerprinting efficiency of an algorithm utilizing only selected features (Table 12). Obtained with the first solution entropy is extraordinarily satisfactory, in fact almost ideal as for the available dataset. Yet, `bf.js` could not be used in a production environment since it was not built with such intention — its execution time is exceedingly high (3.9s) and instability (a change observed each 3.5 days) leaves much to be desired. Nonetheless,

Table 11: Final fingerprinting methods evaluation.

Feature	Verdict	Test signature	codeL	execT	Discussion
addBehaviour	included	addBehavior	50B	1ms	low entropy but stable
ad-block add-on	rejected	-	-	-	rejected due to potential instability
appCodeName	included	appCodeName	50B	1ms	low entropy but stable
appName	included	appName	50B	1ms	low entropy but stable
appVersion	modification	-	100B	2ms	appVersion parts including version strings (e.g. 1.5.11.0) should be trimmed to ensure stability
canvas	included	canvas-advanced-min	900B	6ms	a test with the highest stability, relatively high entropy and quite effective execution time-wise has been chosen
cookies	included	cookies, cookies-active	200B	2ms	low entropy but stable
CPU class	included	cpuClass, cpuClass-oscpu	100B	2ms	low entropy but stable
DNT header	included	dnt-navigator, dnt-window	150B	1ms	
fonts	modification	-	2,5KB	20ms	a test for 100 selected fonts with *monospace* as the base font and size of 70 pixels
indexedDb	included	indexedDb	50B	1ms	
language	included	language, language-browser, language-system, language-user	200B	4ms	
localStorage	included	localStorage, localStorage-active	200B	2ms	
openDb	included	openDatabase	50B	1ms	
screen pixel ratio	included	screenPixelRatio	50B	1ms	
screen resolution	rejected	-	-	-	rejected due to potential instability
screen color depth	included	screenColorDepth	50B	1ms	
sessionStorage	included	sessionStorage, sessionStorage-active	200B	2ms	low entropy but stable
timezone	included	timezone	50B	1ms	
touch support	included	touchSup-basic, touchSup-msPointer	200B	2ms	
User-Agent	modification	-	-	-	User-Agent parts including version strings (e.g. 1.5.11.0) should be trimmed to ensure stability
UA properties	rejected	-	-	-	to save the script code length by not including parser external library, User-Agent approach was chosen
webGL	included	-	-	-	both webGL drawing and properties proved high instability therefore have been rejected

Table 12: Comparison of different solutions.

Solution	dtC	uqC	E(x)	modC	modT	codeL	execT
all `bf.js` fingerprints	4921	4909	12.26	543	3d 12h	85KB	3.9s
final selection of fingerprints	4385	4020	11.96	249	6d 4h	29KB	0.4s
an ideal solution	5038	5038	12.30	0	-	-	-

the production solution, while matching all the expectations listed previously, achieved likewise high diversity — only 0.3 less bits of entropy. The execution time of 0.4s is excellent, the number of changes dropped by a half and the average time distance of a change improved by almost 3 days, which is highly more acceptable.

Server-based solutions. 6 days of fingerprint stability achieved with the proposed production solution is far behind cookie-based identifiers that are able to last for years. The need for more advanced techniques is a natural way of improving the process of fingerprint creation. This work has employed certain aspects of a potential server-based solution, thus few conclusions that could be useful in creating such solutions were summarized.

The primary obstacle is the transfer of obtained in the browser data to a server. Length of certain fingerprints (e.g. canvas, *webGL*) proved to be unacceptable, thus the author suggests compressing the data by applying a hashing algorithm before the transfer. Locality preserving hash could be utilized in case the server logic would implement a tracking of value changes — it would allow to measure the change extent. By having such hashes for the most expensive fingerprints and implementing translation and compression methods for the remaining ones (e.g. true/false setting sent as one bit of information, mapping of common phrases to shorter symbols), the necessity to use *CORS* POST request could be possibly reduced. Because *CORS* introduces a noticeable connection overhead, having a fingerprint compressed enough to fit a GET parameter would significantly advance the performance. The primary obstacle is the transfer of obtained in the browser data to a server. Length of certain fingerprints (e.g. canvas, *webGL*) proved to be unacceptable, thus the author suggests compressing the data by applying a hashing algorithm before the transfer. Locality preserving hash could be utilized in case the server logic would implement a tracking of value changes — it would allow to measure the change extent. By having such hashes for the most expensive fingerprints and implementing translation and compression methods for the remaining ones (e.g. true/false setting sent as one bit of information, mapping of common phrases to shorter symbols), the necessity to use *CORS* POST request could be possibly reduced. Because CORS introduces a noticeable connection overhead, having a fingerprint compressed enough to fit a GET parameter would significantly advance the performance.

To improve the JavaScript code execution time, its length and the size of transferred data, some fingerprints could be processed on the back-end side instead in the user's browser, e.g. User-Agent accessible from HTTP request headers holds identical information as the value returned by JavaScript API — server could utilize parsing libraries to extract meaningful data.

6. Conclusions and Future Work

As digital transactions and interactions continue to grow in volume and importance, the necessity of authentication and verification of the identity of their participants will continue to grow. One of the mechanisms that can be utilized for this purpose is fingerprinting, which plays recently a more and more significant role in sustainable user authentication and web tracking. This techniques has a very broad scope of use cases, among which are fraud detection, adjusting security policies and management, identifying and blocking botnets, real-time target marketing, limiting access to services (for example when filling the surveys), to name just a few. Due to fingerprinting analytics linking a device to a user is possible, as well as identifying the user who uses multiple devices to access the same service.

Despite the indisputable role of computing device fingerprinting in web tracking, this paper proves that the application of fingerprinting technique is really demanding, and it requires a lot of effort to develop an efficient fingerprinting algorithm. The resulting solution presents satisfactory performance in terms of diversity, execution time and the length of the code bundle, yet demonstrates a need for improvement of its stability, which is essential in most of the usages.

Except for the benefits coming from conducting the first evaluation of different fingerprint implementations and producing an optimal set of features, this work allows to draw many additional conclusions. The implication of the performed analysis of existing solutions is the revealment of some misconceptions that they introduce — creating artificial fingerprints like browser tempering is only exacerbating the overall efficiency. Some of the fingerprints (ad-block extension detection, flash-based) have been found to be unstable between regular browsing and private-mode, something that should not make a difference to a respectable algorithm. An instability of certain fingerprints was observed and discussed altogether with potential causes and possible improvements. Finally, the paper proves the superiority of CSS-based font probing over canvas-based solutions and allows to select a reference set of fonts providing the best detection performance. Additionally, some important objectives of an advanced server solution were pointed out. The outcome of our research provides a noticeable progress in the fingerprinting analytics. The discovered features and corresponding optimal implementations will enrich and improve an open-source fingerprinting library *Fingerprintjs2* that is daily consumed by hundreds of websites.

Although this paper presents detailed and interesting results, it suffers also from some limitations. First of all, this research intended to test all the features, which according to the authors knowledge could serve as fingerprints. However, an unexpectedly large number of possible attributes and their modifications have surpassed the authors capabilities. The fact that many additional fingerprinting features have not been evaluated raises a number of opportunities for future research and analysis of the remaining fingerprints. Moreover, certain test outcomes did not allow to perform their full assessment, thus continuation of their evaluation could bring important findings in terms of their usability. Importantly, a short period of data collection, resulting in a decent but limited dataset, did not allow to conclude reliably in a few aspects — following research should be conducted in the long-term to eliminate such concerns.

Device fingerprinting proves to be a powerful technique, yet leaving a large room for improvement. Thus, further research have to be conducted in order to decrease the efficiency distance with well-known storage-based methods. Also, the detailed analysis of numerous possible practical applications of fingerprints, and the description of the proposed scenarios of their usage is assumed.

References

[1] R. Amin, N. Kumar, G. Biswas, R. Iqbal, V. Chang, A light weight authentication protocol for IoT-enabled devices in distributed cloud computing environment, Future Generation Computer Systems.doi:10.1016/j.future.2016.12.028.

[2] V. Chang, Y. Kuo, M. Ramachandran, Cloud computing adoption framework: A security framework for business clouds, Future Generation Comp. Syst. 57 (2016) 24–41. doi:10.1016/j.future.2015.09.031.

[3] G. Sun, D. Liao, H. Li, H. Yu, V. I. Chang, L2P2: A location-label based approach for privacy preserving in LBS, Future Generation Computer Systems. 74 (2017) 375–384. doi:10.1016/j.future.2016.08.023.

[4] D. M. Kristol, HTTP cookies: Standards, privacy, and politics, ACM Trans. Internet Techn. 1 (2) (2001) 151–198. doi:10.1145/502152.502153.

[5] P. Rabinovich, Secure cross-domain cookies for HTTP, J. Internet Services and Applications 4 (1) (2013) 13:1–13:12. doi:10.1186/1869-0238-4-13.

[6] S. Sivakorn, I. Polakis, A. D. Keromytis, The cracked cookie jar: HTTP cookie hijacking and the exposure of private information, in: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, 2016, pp. 724–742. doi:10.1109/SP.2016.49.

[7] P. Eckersley, How unique is your web browser?, in: Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings, 2010, pp. 1–18. doi:10.1007/978-3-642-14527-8_1.

[8] Q. Xu, R. Zheng, W. Saad, Z. Han, Device fingerprinting in wireless networks: Challenges and opportunities, IEEE Communications Surveys and Tutorials 18 (1) (2016) 94–104. doi:10.1109/COMST.2015.2476338. URL https://doi.org/10.1109/COMST.2015.2476338

[9] F. Alaca, P. C. van Oorschot, Device fingerprinting for augmenting web authentication: classification and analysis of methods, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016, 2016, pp. 289–301.

[10] E. Bursztein, A. Malyshev, T. Pietraszek, K. Thomas, Picasso: Lightweight device class fingerprinting for web clients, in: Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM@CCS 2016, Vienna, Austria, October 24, 2016, 2016, pp. 93–102.

[11] Fingerprints2, Fingerprintjs2 - modern browser fingerprinting library. [on-line] https://github.com/valve/fingerprintjs2. (2016).

[12] A. Cahn, S. Alfeld, P. Barford, S. Muthukrishnan, An empirical study of web cookies, in: Proceedings of the 25th International Conference on World Wide Web, WWW '16, 2016, pp. 891–901.

[13] Persistent, Usage of persistent cookies for websites. [on-line] https://w3techs.com/technologies/details/ce-persistentcookies/all/all (retrieved: 08/2016). (2016).

[14] M. E. Whitman, J. Perez, C. Beise, A study of user attitudes toward persistent cookies, Journal of Computer Information Systems 41 (3) (2001) 1–7.

[15] C. Low, Cookie law explained. [on-line] https://www.cookielaw.org/the-cookie-law/ (retrieved:08/2016). (2016).

[16] J. Hayes, 'cookie law': a hostage to fortune?, Engineering Technology 7 (8) (2012) 66–69. doi:10.1049/et.2012.0812.

[17] D. Fetterly, M. Manasse, M. Najork, J. Wiener, A large-scale study of the evolution of web pages, in: Proceedings of the 12th International Conference on World Wide Web, WWW '03, ACM, 2003, pp. 669–678.

[18] M. S. Siddiqui, D. Verma, Evercookies: Extremely persistent cookies, International Journal of Computer Science and Information Security 9 (5) (2011) 165.

[19] G. Goth, Privacy gets a new round of prominence, IEEE Internet Computing 15 (1) (2011) 13–15.

[20] D. Kim, Poster: Detection and prevention of web-based device fingerprinting, in: IEEE Symposium on Security and Privacy (SP), 2014.

[21] T. Yen, Y. Xie, F. Yu, R. P. Yu, M. Abadi, Host fingerprinting and tracking on the web: Privacy and security implications, in: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012, 2012.

[22] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, B. Preneel, Fpdetective: dusting the web for fingerprinters, in: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 1129–1140.

[23] T. D. Laksono, Y. Rosmansyah, B. Dabarsyah, J. U. Choi, Javascript-based device fingerprinting mitigation using personal http proxy, in: Information Technology Systems and Innovation (ICITSI), 2015 International Conference on, IEEE, 2015, pp. 1–6.

[24] P. Baumann, S. Katzenbeisser, M. Stopczynski, E. Tews, Disguised chromium browser: Robust browser, flash and canvas fingerprinting protection, in: Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, ACM, 2016, pp. 37–46.

[25] N. Takei, T. Saito, K. Takasu, T. Yamada, Web browser fingerprinting using only cascading style sheets, in: Broadband and Wireless Computing, Communication and Applications (BWCCA), 2015 10th International Conference on, IEEE, 2015, pp. 57–63.

[26] Rfc 2663 specification — ip network address translator (nat) terminology and considerations, [on-line] `https://tools.ietf.org/html/rfc2663` (Retrieved: 08/2016).

[27] K. Egevang, P. Francis, The ip network address translator (NAT), Tech. rep. (1994).

[28] T.-F. Yen, X. Huang, F. Monrose, M. K. Reiter, Browser fingerprinting from coarse traffic summaries: Techniques and implications, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2009, pp. 157–175.

[29] K. Boda, Á. M. Földes, G. G. Gulyás, S. Imre, User tracking on the web via cross-browser fingerprinting, in: Nordic Conference on Secure IT Systems, Springer, 2011, pp. 31–46.

[30] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, C. Diaz., The web never forgets: Persistent tracking mechanisms in the wild. technical report, princeton university, ku leuven (2014).

[31] S. Englehardt, A. Narayanan., On-line tracking: A 1-million-site measurement and analysis. technical report, princeton university (2016).

[32] A. Narayanan, H. Paskov, N. Z. Gong, J. Bethencourt, E. Stefanov, E. C. R. Shin, D. Song, On the feasibility of internet-scale author identification, in:

Security and Privacy (SP), 2012 IEEE Symposium on, IEEE, 2012, pp. 300–314.

[33] K. Boda, Á. Földes, G. Gulyás, S. Imre, User tracking on the web via cross-browser fingerprinting, Information Security Technology for Applications (2012) 31–46.

[34] C. Ferreira Torres, H. Jonker, S. Mauw, Fp-block: usable web privacy by controlling browser fingerprinting, Computer Security–ESORICS 2015 (2015) 3–19.

[35] N. Nikiforakis, W. Joosen, B. Livshits, Privaricator: Deceiving fingerprinters with little white lies, in: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2015, pp. 820–830.

[36] C. Hothersall-Thomas, S. Maffeis, C. Novakovic, Browseraudit: automated testing of browser security features, in: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ACM, 2015, pp. 37–47.

[37] S. Madan, S. Madan, Security standards perspective to fortify web database applications from code injection attacks, in: Intelligent Systems, Modelling and Simulation (ISMS), 2010 International Conference on, IEEE, 2010, pp. 226–230.

[38] P. Laperdrix, W. Rudametkin, B. Baudry, Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints, in: 37th IEEE Symposium on Security and Privacy (S&P 2016), 2016.

[39] K. Mowery, H. Shacham, Pixel Perfect: Fingerprinting Canvas in HTML5, Tech. rep., University of California, San Diego, USA (2012).

[40] A. Kobusinska, J. Brzezinski, K. Pawulczuk, Device fingerprinting: Analysis of chosen fingerprinting methods, in: Proceedings of the 2nd Interna-

tional Conference on Internet of Things, Big Data and Security, IoTBDS 2017, Porto, Portugal, April 24-26, 2017, 2017, pp. 167–177.

[41] J. Brzezinski, A. Danilecki, M. Holenko, A. Kobusinska, J. Kobusinski, P. Zierhoffer, D-reserve: Distributed reliable service environment, in: Advances in Databases and Information Systems - 16th East European Conference, ADBIS 2012, Poznań, Poland, September 18-21, 2012. Proceedings, 2012, pp. 71–84.

[42] A. Danilecki, M. Holenko, A. Kobusinska, M. Szychowiak, P. Zierhoffer, Reserve service: An approach to increase reliability in service oriented systems, in: Parallel Computing Technologies - 11th International Conference, PaCT 2011, Kazan, Russia, September 19-23, 2011. Proceedings, 2011, pp. 244–256.

[43] A. Kobusinska, D. Wawrzyniak, Replication of recovery log - an approach to enhance SOA reliability, in: Distributed Applications and Interoperable Systems - 15th IFIP WG 6.1 International Conference, DAIS 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings, 2015, pp. 152–157.

[44] C. Maurice, S. Onno, C. Neumann, O. Heen, A. Francillon, Improving 802.11 fingerprinting of similar devices by cooperative fingerprinting, in: SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013, 2013, pp. 379–386.

[45] J. Doherty, Wireless And Mobile Device Security, Jones & Barlett Learning Information Systems Security & Assurance, Jones & Bartlett Learning; 1 edition (January 6, 2015), 2015.

[46] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, G. Vigna, Cookieless monster: Exploring the ecosystem of web-based device fin-

gerprinting, in: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, 2013, pp. 541–555.

[47] BlockAdBlock — AdBlock add-on detection library., [on-line] `https://github.com/sitexw/BlockAdBlock` (Retrieved: 08/2016).

Anna Kobusinska received her M.Sc. and PhD degrees in computer science from Poznań University of Technology, in 1999 and 2006, respectively. She currently works as an Associate Professor at the Laboratory of Computing Systems, Institute of Computing Science, Poznań University of Technology, Poland. Her research interests include large-scale distributed systems, service-oriented and cloud computing. She focuses on distributed algorithms, Big Data analysis, replication and consistency models, as well as fault-tolerance, specifically checkpointing and rollback recovery techniques.

She has served and is currently serving as a PC member of several international conferences and workshops. She is also author and co-author of many publications in high quality peer reviewed international conferences and journals. She participated to various research projects supported by national organizations and by EC in collaboration with academic institutions and industrial partners.

*Biographies (Text)

Jerzy Brzeziński received M.Sc. in electrical engineering, and Ph.D. and Dr. Habil. in computer science, all from Poznań University of Technology, where he is currently a Full Professor of Computer Science.

His research interests include distributed algorithms and fault-tolerant distributed systems. He is the author and coauthor of two books, and over 100 research papers published in journal and proceeding of many international conferences. He has been involved in many international and national research projects.

Prof. Brzeziński is a member of the IEEE CS, ACM, Polish Information Processing Society, Computer Science Committee of the Polish Academy of Sciences, among others.

Kamil Pawluczuk is a 2016 graduate of Poznan University of Technology. His interests focus on practical implementations of cutting-edge solutions in web security. His Master thesis consisted in development of device fingerprinting algorithm ready to meet real-time environments. Currently he is leading creation of progressive fraud detection system at Beta District, Honk Kong bitcoin-oriented buisness.
He is also very active in tech-related communities. He led Web Application Research Group - Poznań, Poland, and (co)organised conferences and workshops.

*Biographies (

*Biographies (Photograph)

***Biographies (Photograph)**