Device Fingerprinting: Analysis of Chosen Fingerprinting Methods

Anna Kobusińska, Jerzy Brzeziński and Kamil Pawulczuk
[1]*Institute of Computing Science, Poznań University of Technology, Piotrowo 3, Poznań, Poland*

Keywords: IoT, Big Data, Fingerprinting, Web Tracking, Security.

Abstract: Device fingerprinting is a modern technique of using available information to distinguish devices. Fingerprinting can be used as a replacement for storing user identifiers in cookies or local storage. In this paper we discover features and corresponding optimal implementations that may enrich and improve an open-source fingerprinting library *Fingerprintjs2* that is daily consumed by hundreds of websites. As a result, the paper provides a noticeable progress in the analysis of fingerprinting solutions.

1 INTRODUCTION

Many on-line business models are based on the necessity of distinguishing one web visitor from another. Thus, web tracking becomes essential to the World Wide Web. HTTP cookies (RFC, 2016),(Cahn et al., 2016) are heavily consumed for this purpose. Once a web page is requested, a cookie containing a unique identifier is stored on the users computer. Such practice is fundamental for many websites to ensure a high level of usability. At the same time, it is exploited by advertising companies to track user interests and hence, increase the probability of purchase by serving personalized offers. 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. Forthcoming laws and directives became a danger for future usage of this storage type. For these reasons, many other alternatives were considered. Various additional storage-based techniques are daily utilized, thanks to the successful adoption of HTML5 specification (HTML5, 2016) that introduced additional APIs e.g. *localStorage* or *indexedDB*.

However, the past decade brought more advanced invention, something that does not leave any data on the user computer — e-fingerprinting. And it is even more powerful than human fingerprinting. When properly executed, the process may stay unnoticeable. By collecting many small pieces of information about the specific device, one can try to distinguish one from another. Nowadays, it is very unlikely that, having a set of random users, their devices, installed software or its setting will not differ in any way. Large competition of hardware producers, daily software updates caused by the need of addressing the latest security threats, or high personalization trends are just a few of the reasons for the devices to differ. That brings an opportunity for fingerprinting. Information such as *User-Agent header, screen resolution, hardware fingerprint* (e.g. audio, canvas) or approximate location based on IP address, once combined together, hold invaluable identification properties. Such data is easily obtainable from JavaScript. Once the user opens a web page having a fingerprinting script attached, a user identifier can be generated. Simple queries to various APIs yield dozens of values which can be considered as fingerprinting features. The simplest solution to get the final user identifier (out of the features vector), is to apply a hash function to all of the information concatenated into one string. If none of the fingerprints have changed over different visits of the user, such hash is not going to differ between consecutive executions of the algorithm. Therefore, it could be treated as an identifier in the same way as cookie identifiers. Depending on the type of used fingerprinting method, such identifier should be called a device, browser or user fingerprint. Nevertheless, device and browser terms are often considered equal due to a small boundary laying in between.

Such fingerprinting scripts are already in use. *Fingerprintjs2* (Fingerprints2, 2016) is an open-source fingerprinting solution which follows exactly the scenario described above. It is used by many, primarily with aim of blocking abusive users. *Augur* (Augur, 2016) is a commercial solution providing device recognition based on a similar concept. Many

Kobusińska, A., Brzeziński, J. and Pawulczuk, K.
Device Fingerprinting: Analysis of Chosen Fingerprinting Methods.
DOI: 10.5220/0006375701670177
In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pages 167-177
ISBN: 978-989-758-245-5

advertising-related companies have already incorporated basic fingerprinting routines into their cookie syncing scripts, which are the backbone of their businesses. All the examples are collecting fingerprints and generating final identifiers on the client side. The most advanced solutions will send the data to the server which will do the job of putting all the information together.

As the need for additional storageless techniques appeared, various of fingerprinting studies have started. Most of them are focused on evaluating another idea which could be turned into additional fingerprint. They usually discuss the issues related to diversity and stability. These are the primary challenges each solution has to face. 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, due to the considerable speed of evolution of the software, hardware and their settings, fingerprints are changing equally, on daily basis. Such changes have to be tracked down and controlled by additional mechanisms, or unstable features have to be classified and excluded from the process.

Stability and diversity are the most important criteria for all of the fingerprint usages, yet many businesses are restricted with additional conditions which this work focuses on. The length of execution code, execution time and the length of the final fingerprint are crucial limitations of any real-time fingerprinting solutions. So far, despite a noticeable need of many companies that are trying to implement early solutions, they were not addressed by other theoretical studies. Thus, this study aims on implementing various methods of fingerprint collection and comparing them accordingly to the most restrictive needs. In the paper, a wide discussion of available fingerprinting methods was conducted. A set of most promising ones was chosen for evaluation and has been implemented within a fingerprinting environment. Developed script has been executed on thousands of different user browsers in order to collect real fingerprinting data. This data has been a subject of excessive analysis. As a result of cost-benefit evaluation, a set of features and respective optimal fingerprinting implementations has been chosen.

The paper is organized as follows. Section 2 describes the topic background: explains web tracking and available methods, introduces the term of fingerprinting, its usages and challenges. Section 3 discusses the literature of the topic. Section 4 presents the architecture developed for the purpose of analysis of various fingerprint features. Next, Section 5 presents the obtained results and their discussion, while the last Section brings final conclusions and proposes the further steps.

2 DEVICE FINGERPRINTING BACKGROUND

Web tracking is commonly known as assigning a unique and possibly stable identifier to each user visiting a website. The general purpose is to connect future page views of the same person or 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 regarding whether it uses any of the storage mechanisms on the client side, i.e. storage-based and storageless techniques.

2.1 Storage-based Techniques

A well known representative of this group are HTTP cookies (Cahn et al., 2016). According to Web Technology Survey statistics (Persistent, 2016), 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 (Low, 2016), 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 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 (Fetterly et al., 2003). 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 similarly by serving files containing variable definitions of unique ids — 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 (Kamkar, 2016). 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.

2.2 Storage-less Techniques

One category of methods which are not employing any storage are state-based techniques, also known as history stealing. Considered as attacks, they 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 hyper-links 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.

Attribute-based and setting-based methods are second half of storageless techniques. They are often referred to as *fingerprinting* (device, browser or user fingerprinting) (Yen et al., 2012), (Acar et al., 2013). Focusing on collecting as many small pieces of information as possible and then putting them together is giving reasonably unique device identification. Various categories of fingerprints could be determined: 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), active (measuring, tracking or active querying in purpose of collecting additional information).

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 (Webkit2016, 2016) 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.

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 (Yen et al., 2009), (Boda et al., 2011). 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.

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:

$$H(X) = -\sum_{i=1..n} P(x_i) * log_2 P(x_i),$$

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 $H(X) \approx log_2|X|$.

3 RELATED WORK

In 2010, EFF published a reference study (Eckersley., 2010) 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 (Acar et al., 2014), 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 (Englehardt and Narayanan., 2016).

There are also plenty of websites aimed at raising awareness of tracking among Internet users. Many on-line fingerprinting tools (Frontier, 2016), (Cross-browser, 2016), (Kurent, 2016), (Tillmann, 2016), 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 (BrowserSpy, 2016), (Checklist, 2016).

4 EMPIRICAL EVALUATION

Analysis environment consists of three parts: fingerprinting script, back-end service and analysis tools. 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 the setup. General process of gathering fingerprint samples is presented in Figure 1. The script was exposed within Amazon S3 Bucket and could be linked to any website. When a user entered one of the collaborating web pages, the script was downloaded and executed as one of the assets. The outcome was sent directly to the study server (Amazon EC2), which processed the data, appended backend-side fingerprints (HTTP request headers) and eventually, stored it into DynamoDB database for further analysis. The statistics were generated with analysis tool that fetched the data directly from Amazon.

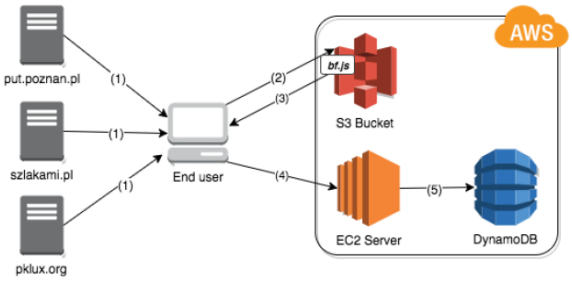

Figure 1: Fingerprinting process scheme.

Fingerprinting script called *bf.js* has been developed. Once triggered, it collects all implemented fingerprints and sends them to the server, where they are stored in the database.

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 1 , 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.

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.

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 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 transfered 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 Amazons 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 / s 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.

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. It provided all the essential information about the work, simultaneously trying to persuade websites administrators to get involved. Obviously, it was not an easy task since foreign script execution may cause a serious damage. Therefore, hosts that have taken part in the study were mostly found as colleagues of the author, except of those, who accepted the petition with a privilege of attaching the script as a local resource (to protect from script modification), after reviewing it. The data have been collected from 7 participating websites during approximate period of one month. In total 15042 records from 5038 users were obtained.

5 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).

5.1 Evaluation Criteria and Data Representation

Except of identification of the best possible fingerprinting implementations of certain features, each attribute has been analyzed according to the following criteria:

Diversity — basic criterion for each fingerprinting study, 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 states how often a fingerprint is changing its value over the time. 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. Thus, length of minified code for each method implemented in *bf.js* was included. Advanced fingerprints rely on time-consuming processing that makes another limitation. Thus, **execution time** has been measured for each method independently so the average time could be calculated.

Length of the Fingerprint — in scenario when all the results are transfered to the server unchanged, their overall size is a shortcoming. Average length of sent data was computed as the last criterion.

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–ids*. 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. 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.

This evaluation, having 5000 samples, could achieve at best $log_2 5000 \approx 12.3 bits$.

5.2 Discussion

The number of possible features to be fingerprinted is immense. This work is focused on browser fingerprinting. 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. 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.

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 it 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. Due 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.

Canvas Fingerprinting. Canvas is an HTML element used to draw basic 2D graphics on a web page. Since this fingerprint was very popular within past years, many different ways of implementation were discovered. In this paper 12 canvas fingerprint tests were collected to answer the question which properties are the most valuable. As a result, the following conclusions were drawn:

- 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.

- Tests for blending and winding support improved the overall result.
- The smile icon rendering test achieved a surprisingly high score of entropy. The most common values in the dataset were following (some of them seem to be identical while there are small differences when compared binary)
- Surprisingly, 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 uniques and distinct values.
- Adding a number to a text increased overall diversity. As the test for special characters was not implemented in a proper way (as extension instead of method replacement), the result does not allow to draw any particular conclusions.

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 source of instability and, at the same time, of entropy. The bigger the canvas and drawn elements are, the higher the entropy, instability and execution time. The only stable element seem 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. Browsers are exposing cookie support setting via *navigator.cookieEnabled* property. 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 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. It confirms that small amount of collected data does not allow to draw widely applicable conclusions.

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*. 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 (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. 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. 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, due 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. 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 plug-ins (*Adobe Flash* of Java), however there are hacks to obtain a partial collection. 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 sample 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* script. The comparison of the observations of each method is the following:

- Average execution time of canvas-based font probing was roughly three times slower.
- CSS detection slightly outranks canvas but in both methods efficiency is almost complete (assessed with manual verification).
- 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 author suspects that CSS methods "reserves" the space (maximal height) for any character supported by a font, also if they are not printed.

- In some browsers discrepancies of 1 pixel were observed. Therefore, the tests were improved to meet this margin of error.
- 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).
- Test string size of 70 pixels produced almost identical results as 180 or 200 pixels.
- *monospace* font was slightly more effective than *sans-serif*, both for CSS and canvas tests.
- The only drawback of CSS method remains the fact that it requires to be executed in users 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 author 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. 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 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.

Language Setting. Exposed by *navigator* object language property, 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 [29]. 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 taking them into consideration.

Platform Fingerprint.*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. Example values are: *Linux aarch64*, *MacIntel*, *iPhone*, *Nokia Series 40* or *PlayStation 4*. This 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. *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. Among both *screenColorDepth* and *screenPixelRatio* tests, stable but rather similar values were collected, providing 0.74 and 0.82

bits 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. 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 didnt change).

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. Timezone fingerprint 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 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 [34] 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. 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. 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 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. 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 Summary

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 a 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.

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.

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, 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 were summarized.

The primary obstacle is the transfer of data obtained in the browser to the 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 networking performance.

To improve the JavaScript code execution time, its length and the size of transfered data, some fingerprints could be processed on the back-end side instead in the users 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

Fingerprinting, as a mechanism used in security and advertisement, plays an important role in web tracking. This work proves that this storage-less technique is really demanding and it requires a lot of effort to develop an efficient fingerprinting algorithm. The resulting solution presented satisfactory performance in terms of diversity, execution time and the length of the code bundle, yet demonstrated 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. Analysis of existing solutions revealed 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, this work 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 this research provides a noticeable progress in the analysis of fingerprinting solutions. The discovered features and corresponding optimal implementations will enrich and improve an open-source fingerprinting library *Fingerprintjs2*.

This study was not able to evaluate many additional features to be fingerprinted, therefore an analysis of remaining ideas could take place. 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. Further researches have to be conducted in order to decrease the efficiency distance with well-known storage-based methods.

REFERENCES

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

Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. (2013). Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM.

Augur (2016). Augur, a set of apis and tools that instantly enables businesses to recognize devices, and consumers across devices. [on-line] https://www.augur.io/ (retrieved: 08/2016).

Boda, K., Földes, Á. M., Gulyás, G. G., and Imre, S. (2011). User tracking on the web via cross-browser fingerprinting. In *Nordic Conference on Secure IT Systems*, pages 31–46. Springer.

BrowserSpy (2016). Browserspy on-line ngerprinting test tool. [on-line] http://browserspy.dk/ (retrieved: 08/2016).

Cahn, A., Alfeld, S., Barford, P., and Muthukrishnan, S. (2016). An empirical study of web cookies. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 891–901.

Checklist, S. (2016). Web browser security checklist. [on-line] https://www.browserleaks.com/ (retrieved: 08/2016).

Cross-browser (2016). Cross-browser ngerprinting test 2.0. [on-line] https://fingerprint.pet-portal.eu/ (retrieved: 08/2016).

Eckersley., P. (2010). How unique is your web browser? in international symposium on privacy enhancing technologies symposium, pages 118. springer, 2010.

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

Fetterly, D., Manasse, M., Najork, M., and Wiener, J. (2003). A large-scale study of the evolution of web pages. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 669–678. ACM.

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

Frontier, E. (2016). On-line ngerprinting test conducted by electronic frontier foundation. [on-line] https://panopticlick.eff.org/ (retrieved: 08/2016).

HTML5 (2016). HTML5, a vocabulary and associated apis for html and xhtml. http://aiweb.techfak. uni-bielefeld.de/content/bworld-robot-control-software/ adsfdf afdfds afsddfs adfd adfdf adfsdfs adfsdsf afsddfs. [on-line] https://www.w3.org/tr/html5/ (retrieved: 08/2016).

Kamkar, S. (2016). Evercookie virtually irrevocable persistent cookies. [on-line] http://samy.pl/evercookie/ (retrieved: 08/2016).

Kurent, A. (2016). Crossbrowser device ngerprinting diploma thesis. [on-line] http://fingerprinting.comyr.com/ (retrieved: 08/2016).

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

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

RFC (2016). RFC 6265 specication. http state management mechanism. [on-line] https://tools.ietf.org/html/rfc6265 (Retrieved: 08/2016).

Tillmann, H. (2016). Browser ngerprinting test by henning tillmann. [on-line] http://bfp.henning-tillmann.de/ (retrieved: 08/2016).

Webkit2016 (2016). Fingerprinting in webkit. [on-line] https://trac.webkit.org/wiki/fingerprinting.

Yen, T.-F., Huang, X., Monrose, F., and Reiter, M. K. (2009). Browser fingerprinting from coarse traffic summaries: Techniques and implications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 157–175. Springer.

Yen, T.-F., Xie, Y., Yu, F., Yu, R. P., and Abadi, M. (2012). Host fingerprinting and tracking on the web: Privacy and security implications. In *NDSS*.