A Collaborative Strategy for mitigating Tracking through Browser Fingerprinting

Alejandro Gómez-Boix, Davide Frey, Yérom-David Bromberg, Benoit Baudry

A Collaborative Strategy for Mitigating Tracking through Browser Fingerprinting

Alejandro Gómez-Boix
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
alejandro.gomez-boix@inria.fr

Davide Frey
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
davide.frey@inria.fr

Yérom-David Bromberg
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
david.bromberg@inria.fr

Benoit Baudry
KTH Royal Institute of Technology
Stockholm, Sweden
baudry@kth.se

ABSTRACT

Browser fingerprinting is a technique that collects information about the browser configuration and the environment in which it is running. This information is so diverse that it can partially or totally identify users online. Over time, several countermeasures have emerged to mitigate tracking through browser fingerprinting. However, these measures do not offer full coverage in terms of privacy protection, as some of them may introduce inconsistencies or unusual behaviors, making these users stand out from the rest.

We address these limitations by proposing a novel approach that minimizes both the identifiability of users and the required changes to browser configuration. To this end, we exploit clustering algorithms to identify the devices that are prone to share the same or similar fingerprints and to provide them with a new non-unique fingerprint. We then use this fingerprint to automatically assemble and run web browsers through virtualization within a docker container. Thus all the devices in the same cluster will end up running a web browser with an indistinguishable and consistent fingerprint.

ACM Reference Format:
Alejandro Gómez-Boix, Davide Frey, Yérom-David Bromberg, and Benoit Baudry. 2019. A Collaborative Strategy for Mitigating Tracking through Browser Fingerprinting. In *6th ACM Workshop on Moving Target Defense (MTD'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338468.3356828

1 INTRODUCTION

The collection and exploitation of data is very popular and almost normal on the Internet today. The motivations for doing so are multiple. The creation of customer profiles and analysis of behaviors to improve sales and market strategies, the customized manipulation of prices and advertising according to the origin of the target, the monitoring of individuals or groups, the monetization of the information collected through its sale to third parties or statistics are among the most outstanding on the Internet. Studies have shown that user tracking keeps increasing among popular websites [5, 6, 12].

In 2010, through the Panopticlick project, Eckersley collected a set of attributes from HTTP headers, JavaScript and the installed plugins [11]. He demonstrated that this information is so diverse that it can be used for building a browser fingerprint. In the data he gathered 83.6% of users were uniquely identifiable, if users had enabled Flash or Java, this number increased to 94.2%. Later in 2016, Laperdrix et al. [18], with the Amiunique website, conducted a study that confirmed Eckersley's findings, as 89.4% of their collected fingerprints were unique. With the introduction of recent web technologies, like the HTML5 canvas element and the WebGL API, a richer browser fingerprint was built. They also demonstrated that mobile fingerprinting is possible, as 81% of fingerprints from mobile devices were unique in their dataset. The latest study in browser fingerprint diversity, performed by Gómez-Boix et al. [14], analyzed more than 2 million fingerprints. The study raises some new questions on this domain, as only 33.6% of the fingerprints collected were unique. However, the authors showed that fingerprints are very fragile, since by changing just one attribute fingerprints have high chances of becoming unique.

The fact that browser fingerprinting does not store any data on users' devices makes this tracking technique difficult to block. In an attempt to hide the true identity of users, several defenses have emerged over time. Some defenses aim at reducing the diversity surface by blocking the access to specific attributes. Others intend to lie by changing real values, either by randomization or by adding noise. Some defenses are limited in scope, since they target a single attribute, then another vector can be used. Others modify fingerprints inconsistently, which can result in a fingerprint that exhibits non-normal behavior, making it stand out from the rest. Finding a defense against browser fingerprints that offers solid protection while maintaining the usability of web browsers can be a challenge.

The strategy to be applied must be carefully designed, as poor performance can lead to an effect contrary to that desired. We propose an approach that aims to target the entire fingerprint. The idea consist in moving the fingerprints of a set of browsers towards a fingerprint that is common to all of them, so we can create a cluster inside which it is not possible to distinguish a specific browser. The approach we propose is based on the identification of similar devices that are prone to share the same fingerprint, and in

MTD'19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6828-5/19/11…$15.00
https://doi.org/10.1145/3338468.3356828

combination with the automatic reconfiguration of devices modify the fingerprint. We aim at preventing remote sites from identifying browsers by proactively modifying web browser configurations so that a large number of browsers end up having the same or indistinguishable fingerprints. We aim at answering the following research questions:

RQ 1. Can we assemble a browser whose configuration is shared by as many users as possible so that they end up with indistinguishable fingerprints?

RQ 2. What is the cost, in terms of user comfort, of assembling such a browsing platform?

We define a strategy based on collaborative mitigation. In a first step we select the attributes of a device that can be reconfigured. Then, we define a metric that can determine the distance between browser fingerprints. The second step exploits clustering algorithms to allow a large number of browsers to self-organize in disjoint groups characterized by similar fingerprints, to take autonomous reconfiguration decisions that increase the similarity among devices that are in the same cluster. Finally, we generate a common browser fingerprint that minimizes the changes made to the configurations of the users involved. Then, we automatically reconfigure the browsers that belong to the same cluster, to let them expose the same fingerprint. We consider this technique as a moving target defense against browser fingerprinting.

The main contributions of this paper are:

- A collaborative defense that reduces the diversity of browser fingerprints and minimizes the number of changes to be applied to obtain a common, unidentifiable fingerprint.
- A measure for determining the identifiability of a set of browser fingerprints.
- A measure to determine the cost of reconfiguring the web browsers of a group of users by targeting a particular configuration.

The paper is organized as follows. Section 2 presents the background and related work on browser fingerprinting, as well as the current defenses against this tracking technique. Section 3 describes our approach, and the three stages that make up our strategy. Section 4 evaluates the performance of the proposed clustering algorithms and the ability of our strategy to assemble the proposed platforms. Then, Section 5 discusses further developments of our strategy, possible threats to validity and the role of web browsers at mitigating browser fingerprinting. Finally, Section 6 concludes this paper.

2 BACKGROUND

In this section we introduce the foundations of browser fingerprinting. Then, we look at the existing solutions to prevent fingerprint tracking and their effectiveness.

2.1 Browser fingerprint diversity

In 2009, Mayer noticed that a browsers could be distinguished with respect to some attributes which value depends on the operating system, the hardware and the browser configuration [19]. Later, in 2010, Eckersley through the Panopticlick project demonstrated that by collecting device-specific information via a script that runs in the browser, it was possible to uniquely identify users [11]. This study gave rise to a new tracking technique known as browser fingerprinting. Browser fingerprinting consists in collecting data regarding the configuration of a user's browser and system when this user visits a website. The Panopticlick website collected almost half a million fingerprints by collecting information from HTTP headers, JavaScript and installed plugins. Eckersley was able to uniquely identify 83.6% of the visitors. This number reached 94% for devices with Flash or Java installed.

Later, in 2016, with the introduction of new technologies and through the AmIUnique website, Laperdrix et al. [18] corroborated Eckersley's findings. Fingerprints collected by AmIUnique exploited the advances of web standards and HTML5 for fingerprinting. These fingerprints, in addition to the attributes collected by Panopticlick, introduced the HTML5 canvas fingerprinting [21] and the WebGL API. By collecting more than 100,000 fingerprints through the AmIUnique website, Laperdrix et al. observed that 89.4% of fingerprints were unique. The study was performed both on mobile devices and desktop machines. They also demonstrated that fingerprinting mobile devices is possible, as 81% of mobile fingerprints were unique. In our recent work [14], we analyzed the effectiveness of browser fingerprinting at identifying a large group of users. We observed that some elements can affect the effectiveness of browser fingerprinting at tracking users as only 33.6% of more than two million fingerprints we analyzed were unique. The study shows that fingerprints tend to be fragile since only by changing the value of one attribute, fingerprints have high chances of becoming unique.

2.2 Browser fingerprinting countermeasures

Over time several countermeasures have emerged. However, finding an absolute approach that can prevent fingerprinting while maintaining the richness of the modern browser is a challenging task. Some authors have classified the defenses against browser fingerprinting according to various criteria [15, 28]. We classify the existing defenses according to the strategy used.

2.2.1 Script blocking. Some browser extensions, although not designed to counteract browser fingerprinting, can prevent the execution of some fingerprinting scripts. Ad-Block Plus [1], Ghostery [2], uBlock [3], Disconnect [4] and Privacy Badger [5] were the he most popular tracker-blocking browser extensions in 2016 [20]. Privacy Badger uses heuristics to dynamically detect tracking behavior by third-party sites. Ghostery and Privacy Badger are tracking blockers that block scripts that have been blacklisted. NoScript [6] is another browser extension that, similar to Ghostery and Privacy Badger, only runs scripts that have been trusted by the user.

2.2.2 Attribute blocking. Some extension were specifically designed for reducing the diversity surface of fingerprints. When blocking the access to specific attributes, the identifying value of the attribute is reduced and therefore the identifying value of the browser fingerprint decreases, giving a less identifiable browser fingerprint

[1] https://adblockplus.org/
[2] https://www.ghostery.com/
[3] https://www.ublock.org/
[4] https://disconnect.me/
[5] https://www.eff.org/fr/privacybadger
[6] http://noscript.net/

as result. Canvas Defender [7] and CanvasBlocker [8] are extensions that block the access to the HTML5 canvas API by hiding the real canvas value. Canvas Defender is an extension compatible with Chrome and Firefox, while CanvasBlocker is only compatible with Firefox. The drawback is that the use of these extensions can be detected. FP-Scanner is a new test suite that can detect if the canvas element has been altered [28]. One of the strongest defenses against browser fingerprinting is the Tor Browser. Tor developers opted for the uniformity strategy: all Tor browsers will exhibit the same fingerprint, this fingerprint is known and easily identifiable [4]. Tor bases its strategy on basically blocking APIs. The most notable ones are the blocking of the Canvas and WebGL API, the complete removal of plugins, the inclusion of a default bundle of fonts and the modification of the user-agent along with HTTP headers. The Tor Browser will always report that the device is on Windows.

2.2.3 Attribute switching. PriVaricator [23] is a solution that applies randomization policies on two properties of the browser. PriVaricator only focuses on the list of plugins and in the HTML offset measurements to prevent JavaScript font probing. User-agent spoofing extensions aim at increasing the anonymity by lying about the user-agent. User Agent Switcher [9] is an extension compatible with Chrome, Opera and Firefox, that brings the ability to switch user-agents. This extension also offers the option to set-up specific URLs, so the extension can just automatically switch user-agents. Random Agent Spoofer [10] is a Firefox extension that offers protection against fingerprinting by switching between a set of predefined device profiles. A profile, in addition to the user-agent, include other attributes, such as the platform, the timezone, and the screen resolution. All profiles exhibited are extracted from real browser configurations and generally updated at each release. This extension also enables blocking advanced fingerprinting techniques, such as canvas, WebGL or WebRTC fingerprinting. Ultimate User Agent [11] is a Chrome extension that switches the browser's user-agent. By changing the user-agent in the HTTP requests, this extension gives browsers access to websites that demand a specific browser. FP-Block [26] is a browser extension that aims at preserving privacy on the web with respect to browser fingerprinting. FP-Block ensures that two generated fingerprints are sufficiently different to avoid being linked by fingerprint tools. This tool, contrary to naive techniques that mostly randomize the value of attributes, aims at preserving fingerprint consistency.

2.2.4 Attribute blurring. As an extension of attribute switching, another strategy consists in changing the attribute values by adding noise to those attributes that are the result of some rendering process, such as canvas and audio elements. Canvas Defender is an extension that in addition to blocking access to the canvas, has another option in which it adds noise to the canvas element. FaizKhademi et al. [13] created a solution called FPGuard. This solution is based in two phases: runtime fingerprinting detection and prevention. For detecting fingerprinting activities they used a set of nine metrics. For preventing, they applied randomization policies that implied blocking, switching and noise techniques. They target a specific set of attributes, concretely attributes related to the *navigator* and *screen* objects, the list of plugins, the fonts and the canvas rendering. FPRandom [16] is a solution that works at the browser level. It is a modified version of Firefox that adds noise into the canvas and audio elements, and randomizes the enumeration order of JavaScript objects. FPRandom presents two strategies: *Random mode* and *Per session*. In the first mode at every call the browser will return random values, meanwhile in the second one all variables take a random value at startup, remaining over a session.

2.2.5 Reconfiguration. Baumann et al. [8] designed a defense called Disguised Chromium Browser that changes the screen resolution, the browser language, the user-agent, the time and date, the list of fonts and the list of plugins. This solution presents two strategies. In the first one, a set of devices will adopt the same fingerprint for a given period of time. In the second one, the browser configuration is changed at each session. This approach provides consistency in the same browsing session as they use a random session identifier generated at startup to track the modifications made. Blink [17] presents a defense strategy against browser fingerprinting that aims at breaking fingerprint stability over time. Through virtualization, Blink assembles random browser platforms. Contrary to countermeasures that lie on their identity by altering the values of the attributes, Blink assembles genuine platforms that exhibit consistent fingerprints. Blink operates at different levels, from the operating system to the web browser, including fonts and plugins. The downside of Blink is that it can affect significantly the user comfort. Another strategy could consist in alternating between two browser. Although the configurations of the web browsers are not changed, the users would have two distinct device fingerprints. CloakX [27] is a client-side antifingerprinting countermeasure that uses diversification to prevent extension detection and to reduce the accuracy of extension detection. CloakX rewrites the extensions on the client-side and maintains equivalent functionality.

2.3 Effectiveness of countermeasures

Countermeasures for mitigating browser fingerprinting are designed to exhibit different values than the real ones. Most of the approaches presented in the literature are limited in scope. Some countermeasures only target a single feature (e.g. Canvas Defender targets the canvas element) or a set of very limited features (e.g. PriVaricator only targets the list of plugins and the HTML offset measurements). Other approaches have an effect that is opposite to the one intended. User-agent spoofing extensions do not completely cover the navigator object, so they can introduce some degree of inconsistency in the fingerprints (e.g. the information of the OS family exhibited by the user-agent is expected to be consistent with the `navigator.platform`). Nikiforakis et al. [24] demonstrated that incoherent fingerprints make users more distinguishable from the rest of the users, since they exhibit a behavior that is not observed in the wild. Introducing randomization as a defense strategy brings with it the introduction of inconsistency, since there is no compatibility check.

Some approaches make fingerprints stand out from the rest by changing the values of some features: by blocking access, by adding noise or by altering them. For example, preventing the canvas image

[7] `https://multilogin.com/canvas-defender/`
[8] `https://github.com/kkapsner/CanvasBlocker`
[9] `http://useragentswitcher.org/`
[10] `https://github.com/dillbyrne/random-agent-spoofer`
[11] `http://iblogbox.com/chrome/useragent`

from loading is an identifier in itself. Changing fingerprint during a session is also an unusual behavior. Previous studies [24, 25] have demonstrated that it is possible to identify altered browser fingerprints. More recently Vastel et al. [28] demonstrated that it is also possible to uncover the original values that were altered.

Boda et al. [9] proposed a browser-independent fingerprinting method that relies on OS-specific data, fonts, timezone and screen resolution to create an identifier. The study is limited by the size of the analyzed sample. Cao et al. [10] proposed a method for tracking users through cross-browser fingerprinting that relies on operating system and hardware features, such as graphic cards or CPUs. A drawback of this approach is that their whole suite of tests take several seconds to be fully executed.

While Tor Browser offers one of the strongest defenses against browser fingerprinting, it still presents some vulnerabilities. Wang and Goldberg [29] showed that the Tor network is not immune to threats or attackers. Because Tor exhibits a unique fingerprint, its personalization is severely inhibited. A simple change like resizing the browser window makes the browser stand out from the rest of Tor users and can be immediately identified by fingerprinters.

3 APPROACH

The key intuition of our work is as follows: if we can move the fingerprints of a set of browsers towards a fingerprint that is common to all of them, we can create a cluster inside which it is not possible to distinguish a specific browser. To reduce the effectiveness of tracking through browser fingerprinting, we propose an approach that generalizes and automates to some extent the strategy used by the Random Agent Spoofer extension presented above. Our idea is to identify a group of browsers that have similar fingerprints and to collectively move them to a common fingerprint. This section explains how we identify browsers with similar fingerprints, and therefore amenable to collective reconfiguration.

3.1 Collaborative strategy

We propose an approach based on three stages. The first stage consists in selecting those components (attributes) of the web browsers that can be modified without affecting the performance of users while surfing the Internet. In the second stage, we cluster the fingerprints of a population of browsers to identify disjoint groups characterized by similar fingerprints. In the third stage, for each cluster resulting from the previous stage, a set of fingerprints will be created that resembles as closely as possible all the fingerprints in the cluster. Figure 1 summarizes our approach to analyse a population of browsers and perform our moving target defense. Our approach is characterized by three essential properties: (i) the proposed configurations always exhibit consistent fingerprints; (ii) we propose correct configurations of web browsers; and (iii) the fingerprints associated to the proposed configurations shall not be linked to a single device.

3.2 Selection of reconfigurable components

This stage takes as input a set of browser fingerprints. The objective is to select the parts of a device that can be reconfigured without altering the performance of web browsers and without affecting the user comfort.

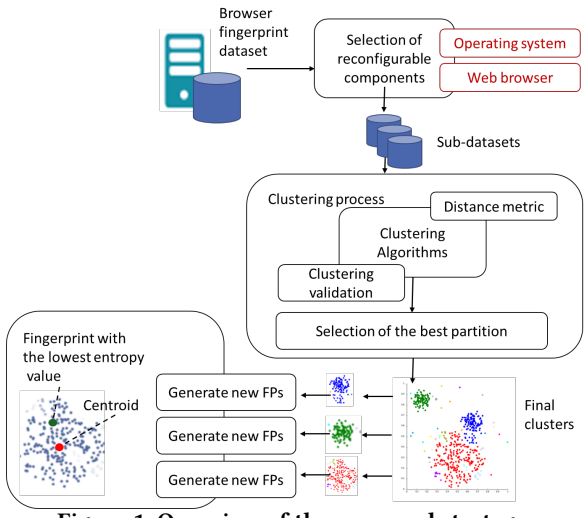


Figure 1: Overview of the proposed strategy.

Analysis of components. We focus the analysis on the components of web browsers that we studied as part of our previous work [14, 18]. Laperdrix et al. [17] classified the browsing components as *alterable* and *essential*. This classification is subjective being related to functional reasons or to comfort. We maintain this basic principle, which will help us in the analysis of the components. For our purpose we classify attributes in *configurable* or *non-configurable*. Below we classify the components and we justify this classification.

Configurable components:

- *Use of an ad blocker*, *Cookies enabled*, *Do Not Track* and *Use of local/session storage* are limited to "yes" or "no", they do not offer very discriminant information, but still can be handle by the users.
- *Content language* and *List of plugins* are user's choice.
- *Timezone*, *screen resolution* and *list of fonts* are attributes indirectly impacted by the environment, even so, their values can be changed.

Non-configurable components:

- *User-agent*, *Platform*, *WebGL Vendor* and *WebGL Renderer* give information about the operating system and hardware.
- *Header-accept*, *List of HTTP headers* and *Content encoding* are related to the HTTP protocol.
- *Canvas* is a dynamic value collectable only at runtime.

Even if the canvas element can be modified, we do not classify it as configurable since a modification of the canvas can be detected as inconsistent. It is similar with the user-agent. Forging user-agents that are consistent is very difficult. The consistency of the information obtained through the user-agent can be verified through other components [24, 28] (e.g. the user-agent is available both from the client side, through the navigator object `navigator.userAgent`, and from the server side, as an HTTP header `User-Agent`). Changing the user-agent without lying implies changing the operating system or the web browser. For reasons of user comfort it is not recommended to change the operating system or the web browser. Consequently, we propose a hard constraint for our approach, which will result in splitting the set of fingerprints in subsets with the

same operating system and web browsers. As browsers are massive pieces of software, by grouping devices with the same operating system and web browser, this approach prevents the introduction of mismatches at the operating system and web browser level. We also affirm that maintaining the operating system and the web browser is a key component so as not to affect the comfort of the users. This means that comparing fingerprints with different operating system or web browser does not make sense. The resulting sets of fingerprints will go through the next stages independently.

After selecting all configurable components, the population of browsers is partitioned into sub-datasets according to the combination of operating system and web browsers (browser versions are not taken into account) [12].

3.3 Clustering process

From the process of selection of reconfigurable components we obtained several datasets of browser fingerprints, the rest of the strategy is applied to each dataset independently. The objective of this stage is to identify similar devices that are prone to share the same fingerprint. Thus, through clustering algorithms a large number of devices will organize themselves into disjointed groups characterized by similar fingerprints.

3.3.1 Dissimilarity metric. In order to identify similar fingerprints, it is necessary to quantify the differences between two fingerprints. To the best of our knowledge, only Laperdrix et al. [17] established a distance function for fingerprints comparison. The distance function involves the eight attributes observed by [11]. We extend this distance function in order to compare the attributes based on their domain.

A distance function is composed of three elements (see equation 1): local distance functions for attribute comparison, weight values for indicating the level of importance for each of the attributes (sum of all weights must be equal to one), and a global function that unifies the local distances and their respective weights.

$$D(F_1, F_2) = \sum_{i=1}^{n} w_i * d\left(F_{1a_i}, F_{2a_i}\right) \tag{1}$$

The local distances and attribute domain are defined in Appendix A. The distance is defined in the range $[0, 1]$. Each attribute is weighted (see https://profileswitch.github.io/ [13]) according to the entropy values obtained in the study performed by Gómez-Boix et al. [14]. Heavier weights indicate more revealing attributes.

3.3.2 Clustering algorithms. In order to obtain sets of browser fingerprints with similar behaviors we apply clustering algorithms based on distance functions. We focus our attention on three well known clustering algorithms based on distance functions: K-Means Clustering, Density Based Clustering Algorithm and Agglomerative Hierarchical Clustering (hereinafter referred to as *KMeans*, *Density-Clustering* and *HierarchicalClustering* respectively). Our main goal is to maximize privacy among devices, in other words, to minimize the identifiability of devices. The clustering process will focus on finding a partition of the dataset that minimizes the identifiability of devices. The lower the number of devices per cluster, the easier it will be to identify them. We formulated Equation 2 for measuring identifiability according to a partition $C = \{c_1, ..., c_K\}$ of the dataset.

$$I = \sum_{k=1}^{K} \frac{1}{u(c_k)} \tag{2}$$

where K is the number of partitions or clusters and $u(c_k)$ is the number of devices in the cluster k. I takes its maximum value when all clusters contain an unique fingerprint, which means that all devices are unique. In this case $I = U$, where U is the total number of devices in the data. I takes its minimum value when all clusters reach the maximum number of devices $\forall c \in C : u(c) = U/K$, meaning that $I = K^2/U$, so I is bounded in the interval $[K^2/U, U]$. When the identifiability values are close to their maximum this means that the clusters obtained contain few devices, easing the total or partial identification of the devices involved. Therefore, the higher the identifiability, the lower the privacy. On the other hand, when the identifiability values are close to their minimum this means that the clusters obtained are close to contain as many devices as possible per cluster, making it difficult to uniquely identify them. Consequently, the lowest the identifiability, the higher the privacy.

The reconfiguration of web browser to adopt a new configuration has an associated cost. So, in addition to minimizing their identifiability, we seek to minimize the number of changes in the configuration of the web browsers of the devices concerned. In order to quantify the level of disruption on a partition of the dataset, we propose Equation 3. To estimate the disruption, we consider two elements: the number of changes made to the fingerprints and the number of devices in each fingerprint. As an approximation of the number of changes, we compute the sum of the distances of all the fingerprints within cluster to the target fingerprint.

$$R = \sum_{k=1}^{K} \sum_{i \in c_k} \left(d\left(FP_i, FP^{\{k\}}\right) * \frac{u(FP_i)}{U} \right) \tag{3}$$

The distance between a given fingerprint and the target fingerprint is weighted by the number of devices in the fingerprint in question. Equation 3 measures the disruption generated by a partition $C = \{c_1, ..., c_k\}$ on the dataset where $u(FP_i)$ is the number of devices who exhibit the fingerprint i. The lower the disruption, the smaller the number of modifications to be made to the configurations of a group of devices in order to reach a given configuration. After executing the clustering algorithm, for each dataset we will obtain disjointed sets of fingerprints, in such a way that optimal values of identifiability and similarity are reached among the fingerprints belonging to the same cluster.

3.4 Fingerprint generation process

This stage performs the moving target defense that aims at "creating" attack-resistant browser fingerprints. For each cluster resulting from the previous stage a set of fingerprints will be created. The main idea consists in proposing a set of configurations that increase the similarity among devices that are in the same cluster. Proposed fingerprints will be shared by all devices within the cluster in order

[12] Operating system and web browser were extract from the user-agent with the Java implementation of ua-parser (https://github.com/ua-parser)

[13] It also contains the algorithms, tools and results used in this study.

to minimize disruption, which means proposing as few changes as possible.

Generating fingerprints. We propose two different alternatives for generating fingerprints:

(1) Taking the cluster centroid.
(2) Taking the fingerprint with the lowest entropy value within the cluster.

By taking the cluster centroid, we ensure that all the fingerprints within the cluster will experience the lowest amount of changes. However, minimizing the changes for fingerprints does not necessarily minimize the number of devices affected. On the other hand, taking the fingerprints with the lowest entropy value within a cluster may cause an increase in the number of changes, but it ensures that users will take a configuration that is most likely to appear in the wild. In principle, we could also consider a third method for generating a new browser fingerprint: assembling a new fingerprint by selecting the values of the components with the lowest entropy values, i.e., the values most likely to appear in the wild individually. But this method would run the risk of assembling wrong or inconsistent fingerprints. In consequence, we do not consider this option.

As result of this stage we obtain a browser fingerprint that is more likely to resist fingerprinting attacks. In addition, the users who decide to adopt this configuration will experience as few changes as possible to their web browser.

4 EXPERIMENTAL ANALYSIS

In this section we present a series of experiment to evaluate the performance of different clustering algorithms and the ability of our strategy at assembling the proposed platforms.

4.1 Research questions

These experiments aim at answering our research questions, which evaluate the effectiveness of our strategy.

RQ1. Can we assemble a browser whose configuration is shared by as many devices as possible so that they end up with indistinguishable fingerprints? This question assesses the ability of our approach at creating common fingerprints, in such a way that their identifiability is reduced as much as possible.

RQ2. What is the cost, in terms of user comfort, of assembling such a browsing platform? This question evaluates whether the number of changes proposed to a group of devices is small enough not to affect user comfort. We quantify the number of changes that are made to the configuration of a set of devices to achieve a common configuration.

4.2 Experiment setup

Finding a partition that maximizes the similarity among fingerprints within the clusters guarantees the success of our approach. Selecting a clustering algorithm that obtains optimal results is crucial. As we mentioned above, we focus our attention on three clustering algorithms: K-Means Clustering, Density Based Clustering Algorithm and Agglomerative Hierarchical Clustering. We use algorithms implemented by the WEKA Workbench [31].

Determining the number of partitions or clusters in a dataset is a problem associated with data clustering, and it is generally different from solving the clustering problem itself. To solve this problem, we propose a binary search to find an optimal number of clusters k. In order to minimize identifiability, we limit the search to the interval $[2, U / 50]$. We empirically assume that if a cluster contains at least 50 devices, the number of devices in the cluster is sufficient so that they cannot be uniquely identified. In our study we set a minimum threshold of 50 devices per cluster. The search aims at minimizing the identifiability of devices, denoted by I (see equation 2). We minimize identifiability while reducing the amount of changes on the configuration of web browsers. For quantifying the level of disruption on a partition of the dataset we use Equation 3.

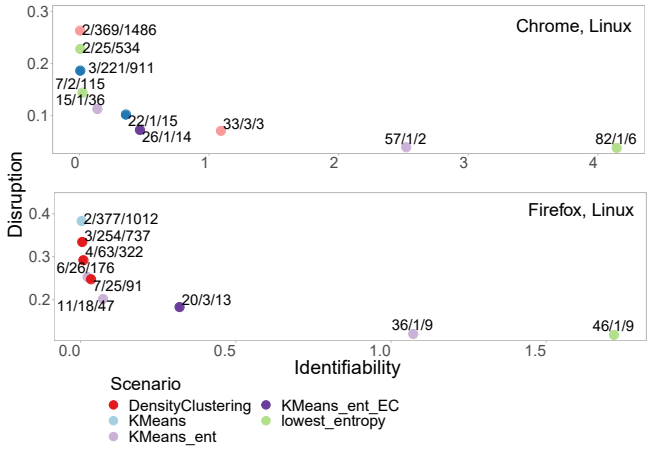
The entropy of Shannon has been used to quantify the level of identifying information in a fingerprint [11, 14, 18]: the smaller the entropy, the less identifiable and the more common the fingerprint. Based on this, we also propose to modify the K-Means Clustering algorithm. The K-Means Clustering algorithm takes random initial centroids, instead we propose to take the fingerprints with the lowest entropy value as the new search start points (hereinafter we refer to this approach as *KMeans_ent*). In addition, we propose to cluster the fingerprints around given initial centroids in two different ways. In the first case we take k random centroids (hereinafter referred to as *KRandom*), and in the second one we choose as centroids the k fingerprints with the lowest entropy values (hereinafter referred to as *lowest_entropy*).

For our experiments we used the dataset that we presented in [14]. As we mentioned, as a result of the first stage we get different subdatasets in which all fingerprints exhibit the same operating system and web browser. 'Dataset' column of Table 1 contains the subdatasets (each dataset represents a web browser and the operating system on which it is running) that we analyzed in our experiments. The column '# of FPs' contains the number of fingerprints that each dataset contains. Since several devices can display the same fingerprint the column 'Number of devices' contains the total number of devices of each data set.

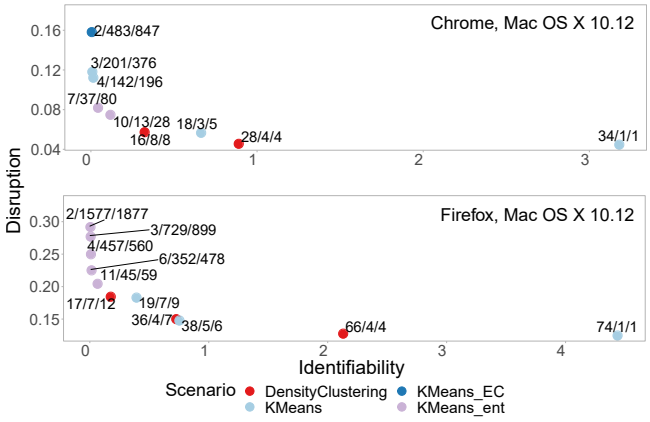
4.3 Results

We call 'scenario' the process of clustering fingerprints in a dataset by performing a binary search through a given clustering algorithm. Figures 2a and 2b show identifiability against disruption resulting from modeling the proposed scenarios in the web browsers Chrome and Firefox, on Linux and Mac OS X 10.12, respectively. The x-axis measures the identifiability for a partition of the dataset. The y-axis measures the disruption caused by a partition of the dataset, taking as target the centroid of each cluster. Each point represents an execution resulting from a clustering algorithm. Clustering algorithms are represented by colors. The labels on the dots contain three elements: the number of clusters (k), the number of fingerprints contained in the smallest cluster and the number of devices contained in the smallest cluster. The suffix "_EC" refers to the second method of generating fingerprints: fingerprints with the lowest entropy values. Both methods of generating fingerprints are based on the same clustering process. So, targeting different fingerprints within the cluster does not change the identifiability measurement, since this measure only takes into account the number of devices

per cluster. Because we aim at minimizing identifiability and disruption, Figures 2a and 2b show the Pareto front. The Pareto front contains only those points which are not dominated by any other point. A point dominates another point if it is better in all relevant dimensions and strictly better in at least one dimension.



(a)



(b)

Figure 2: Measurements of identifiability and disruptions resulting from the modeling of the different scenarios on the Chrome and Firefox web browsers, under Linux (a) and Mac OS X 10.12 (b).

Looking at the identifiability measurements of the raw data (see column *Ident.* in Table 1) and at the figures 2a and 2b we can make the following observations. First, there is no algorithm that performs better in all executions than the others, since the Pareto front is composed of different algorithms. Second, in all scenarios the identifiability of the fingerprints is drastically reduced with respect to the raw data (see column *Ident.* in Table 1). Despite the drastic reduction in identifiability, the main objectives of our strategy are not achieved. This is because (i) there are still clusters with fewer fingerprints than the set threshold and because (ii) uniqueness is not removed at all, as clusters were obtained with a single unique fingerprint. For example, if we analyze the results obtained on Linux (see Figure 2a), the Pareto front for Chrome contains 15 points, when the number of clusters is equal to or greater than 11, all the solutions obtained are non-optimal, resulting in seven non-optimal solutions. All partitions or clustering processes that obtain clusters with fewer devices than the established threshold are considered non-optimal solutions.

Non-optimal solutions tend to appear when the number of clusters approaches $k = U / 50$. Still, these solutions are interesting as they obtain the lowest disruption values of each scenario. In fact, all the scenarios present the same behavior: the greater the number of clusters, the greater the identifiability but the less the disruption, and vice versa, the less the number of clusters, the less the identifiability but the greater the disruption.

Dataset	# of FPs	Number of devices	% of unique	Ident.
Linux, Chrome	1,176	4,117	67.6	921.05
Linux, Firefox	804	2,316	61.3	594.31
Mac OS X 10.12, Chrome	1,047	1,769	73.6	871.24
Mac OS X 10.12, Firefox	3,202	3,832	88.6	2,991.81

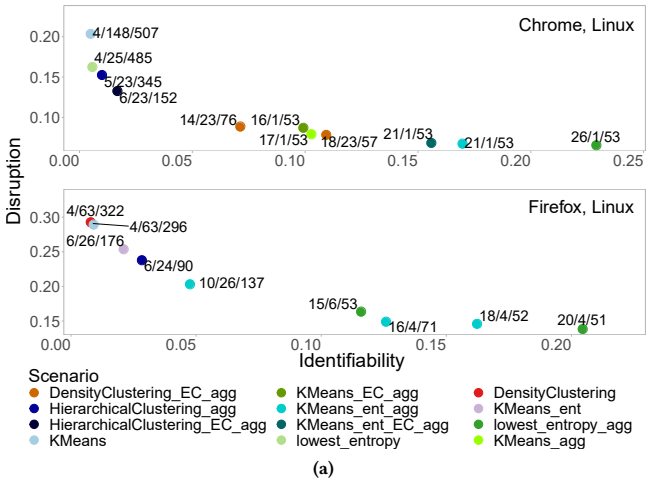
Table 1: Measures of identifiability of raw data.

In order to deal with this issue and achieve optimal solutions we performed an aggregation process on the non-optimal solutions. The process consists of identifying those clusters with less than 50 devices (threshold set for our experiments) for later adding all the fingerprints belonging to these clusters to the nearest cluster with more than 50 devices. To evaluate the performance of the aggregation process, we conducted a second experiment. We computed the number of clusters resulting from the aggregation process and performed a second clustering process with these values. This means that in addition to the k values obtained from the binary search, we extend the search by running the clustering algorithms with these k values. Figures 3a and 3b show identifiability against disruption resulting from the aggregation process and the second clustering process. We refer to the scenarios in which an aggregation process was carried out using the suffix "_agg". Non-optimal solutions were not taken into account for building the Pareto front, even if they presented lower disruption values than the optimal solutions.

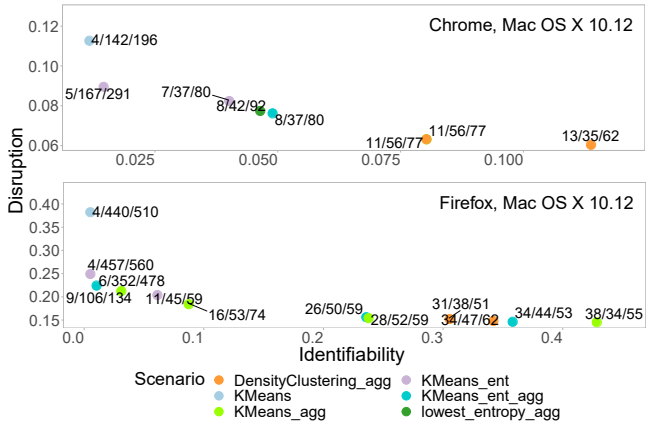
Looking at Figures 3a and 3b we can observe that the Pareto front is composed in great majority of algorithms in which an aggregation process was carried out. If we look at the identifiability values, the aggregation process decreases these values even more.

The nearest point to the origin contains a partition of the data in which both identifiability and disruption are minimized at the most. If we look at Linux (see Figure 3a), for example, for Chrome that point is obtained by the *DensityClustering_EC_agg* algorithm when $k = 14$, and for Firefox it is obtained by the *KMeans_ent_agg* algorithm when $k = 16$. All the optimal solution in the Pareto front reduce drastically the identifiability with respect to the raw data, so generating fingerprints taking as base any of these partitions will meaningfully improve the privacy among devices. However, some solutions (e.g. only two or three clusters) would bring us closer to something like Tor, in which case the diversity would be completely lost. Comparing the solution with the greatest number of clusters and "the best solution" (the nearest point to the origin), we can appreciate that the difference in terms of identifiability is not significant. Taking the solution with the greatest number of clusters

reduces to the most the disruption, being the most flexible solution by offering as many configurations as possible. From now on, we will focus our analysis in the solutions with the greatest number of clusters, which does not mean that the rest of the solutions are not feasible.



(a)



(b)

Figure 3: Measures of identifiability and disruptions resulting from the aggregation process on Chrome and Firefox web browsers, under Linux (a) and Mac OS X 10.12 (b).

Table 2 shows the algorithms with the best performance after the aggregation process with the aim of minimizing disruption. For each dataset we scored the best three algorithms based on the lowest disruption values where the value (1) is the algorithm with the lowest disruption value, so it has the best performance. The ranking aims to minimize disruptions, since the identifiability fell into values low enough that the devices cannot be uniquely identified. Looking at the results shown by table 2, we observe that the best scores are obtained after the aggregation process. Even though *DensityClustering* after aggregating gets slightly better results, with our experiments, we cannot affirm that this algorithm will get the best results in most datasets. Because modeling all scenarios is highly costly in time and resources, selecting the algorithm with

OS	Web Browser	Kmeans	Kmeans_agg	KMeans_ent	KMeans_ent_agg	Kmeans_EC_agg	KMeans_ent_EC_agg	DensityClustering	DensityClustering_agg	lowest_entropy	lowest_entropy_agg
Linux	Chrome				2		3				1
Linux	Firefox				2,3						1
Mac OS X 10.10	Chrome			3		2			1		
Mac OS X 10.10	Firefox	3						1,2			
Mac OS X 10.10	Safari			3					1,2		
Mac OS X 10.12	Chrome				3				1,2		
Mac OS X 10.12	Firefox		1		2			3			
Mac OS X 10.9	Chrome						1				
Mac OS X 10.9	Firefox	3	2								1
Mac OS X 10.9	Safari	2							1,3		
Ubuntu	Chromium				1		2				
Ubuntu	Firefox									2,3	1

Table 2: Ranking of the best algorithms after the aggregation process with the aim of minimizing disruption.

the best performance cannot be done decisively. Looking at Table 2 the three algorithms with the best overall performance were *DensityClustering*, *lowest_entropy* and *KMeans_ent*.

Below is a summary of the steps to take to concretely implement our strategy. In the first stage, those components of the browser on which the strategy will be applied must be identified. Analysis that we performed, in Section 3.2, on the 17 components that compose the fingerprints analyzed in our study. As a result, the distance function will only take into account components marked as configurable. Prior the clustering process, which is the second stage, it is necessary to identify the minimum number of devices per cluster that best suits the total number of devices on which the strategy will be applied (in our experiments we set the threshold to 50). Based on that, the number of cluster k to obtain is equal to the total number of devices divided by the threshold fixed. We recommend running the three algorithms with the best overall performance: *DensityClustering*, *lowest_entropy* and *KMeans_ent* and if there is any non-optimal solution, the aggregation process will be applied to those solutions. From the set of all optimal solution resulting, the partition that minimizes disruption will be selected for generating the new fingerprints. The third stage consists of identifying for each cluster the fingerprint with the lowest entropy value, it means to select the fingerprint with the highest number of devices. This stage has as result a set of fingerprints (configurations). Each device will adopt the configuration of the nearest fingerprint to its fingerprint.

RQ 1.Can we assemble a browser whose configuration is shared by as many users as possible so that they end up with indistinguishable fingerprints? The answer to this question can be found in the set of browser fingerprints generated. We start from the fact that in any scenario where the unique fingerprints are removed or the fingerprints with a small number of devices are reduced, the identifiability decreases.

In Figure 3 we show that all solutions obtained after the aggregation process are optimal, this means that all resulting clusters have a minimum threshold of devices. For each cluster we have two methods to propose the configurations that the devices will adopt. We recommend to adopt the configurations of the fingerprints with the

lowest entropy values over taking the cluster centroids. Adopting fingerprints with the lowest entropy values offers certain advantages. These fingerprints owe the low entropy values to the fact that a large number of devices exhibit the same configuration, this ensures that these fingerprints are the least identifiable within each cluster. For this reason these fingerprints are easily found in the wild. Because they are extracted from correct configurations of web browsers they always exhibit consistent fingerprints. Therefore, as more configurations move toward these configurations, our goal of reducing the fingerprint diversity surface is achieved. Our approach ensures that if all devices within a cluster adopt the recommended fingerprint, they will share an indistinguishable fingerprint.

RQ 2. What is the cost, in terms of user comfort, of assembling such a browsing platform? The answer to this question is closely related to the **RQ1**. Before answering this question, let's analyze two cases that will serve as a reference. The first one is when we move all the fingerprints within the data to a single configuration. In this case, the disruption gets its maximum value, since all devices will change their configuration, however, the identifiability will be zero since there is no diversity. The second case is when all devices show unique fingerprints, in this scenario the identifiability gets its maximum value since all devices are uniquely identifiable and the disruption is zero since no device will change its configuration. Finding the balance between privacy benefit and disruption caused to all users is the key element.

The recommended fingerprints come from clusters resulting from a clustering algorithm in which (i) the number of clusters k is the maximum possible and (ii) an aggregations process was performed ensuring that all solutions are optimal. As we stated, when the number of clusters increases, the identifiability increases while the disruption decreases, so the resulting clusters have the lowest overall disruption value of all optimal solutions. Each method of generating fingerprints has a different impact on the disruption within the cluster. The cluster centroid minimizes the sum of the distances from it to the rest. However, this does not take into account the number of devices that have these fingerprints. With the cluster centroid we take the risk of moving all devices within the cluster to a configuration that is not common, or even worse, that is unique. In contrast, if we take the fingerprint with the lowest entropy value there are no guarantees that the distance from the rest of the fingerprint to it is the minimum, but we certainly know that within the cluster as few devices as possible will change their configuration. By recommending the fingerprint with the lowest entropy value within the cluster we ensure a configuration whose identifiability value is the lowest possible, while at the same time we guarantee to change the configuration to the lowest possible number of devices.

5 DISCUSSION

5.1 Further development

Our strategy aims at assembling browsing platforms that exhibit non-unique fingerprints. These platforms will exhibit consistent fingerprint that will be shared by as many devices as possible. This can be seen as a recommendation system to diminish the identifiability surface of a set of browser fingerprints. Some studies have emphasized the importance of differentiating mobile fingerprints from desktop/laptop fingerprints. This can be explained by the fact that the software environments of mobile devices are much more constrained than on desktop and laptop machines. Due to the limited number of options when customizing mobile browsers, we recommend the use of our strategy on desktop/laptop machines.

For implementing this work in practice, certain issues must be addressed. Once we have identified an appropriate fingerprint (or possibly a set of fingerprints) for the devices within a cluster, we need to find a way to apply it to users' browsing platforms. A simple solution would be to ask users to manually reconfigure their browsers and computers, but this solution would be particularly cumbersome, inconvenient and error-prone. We would have no guarantee that two different users would actually configure the browsers with the same settings, since a browsing platform involves three levels of configurations: hardware, operating system and web browser. Boda et al. [9] showed that gathering enough information from the system it can be still fingerprinted and tracked despite the use of multiple web browsers. For this reason, it is necessary to isolate the device's system from the assembled browsing platform. With this we guarantee the same basis for all devices who share the same operating system. We therefore offer an automated solution that offers each user a new browser that runs locally, on the cloud or on a server chosen by the user. This web browser is configured to provide the selected fingerprint and can be viewed by the user as a normal application through a VNC-based deployment (https://www.realvnc.com/en/). The user therefore would just see an application that differs slightly from the one he or she normally uses.

From a practical point of view, users connect with their own browser to the website with our fingerprint protection platform. The platform collects the fingerprint, and returns a recommended browser fingerprint. In background, the system performs the calculation of clusters on a daily basis and identifies the new fingerprint to be used for each user. The system prepares a docker container with the associated fingerprint and makes it available to the user. The user can then run the container locally, on the platform, or on his own personal server. There is not yet an implementation to our system that executes the complete strategy. We are currently working on the implementation of this proposal on a website https://profileswitch.github.io/. We have a prototype of a web page that collects browser fingerprints and provide each user with a recommended fingerprint. Then we provide a tool that will allow them to run a browser with a standard configuration as close as possible to the one they usually use.

Implementing our proposal through the execution of Docker containers in the user's machine is certainly efficient, lightweight and fast in execution time. While containers can share the operating system kernel with other containers, full virtualized systems are an abstraction of physical hardware that includes a full copy of an operating system, the application and the the dependencies for that application to run. Unfortunately with Docker less isolation is obtained, so running the web browser in a Docker container locally, in theory, does not protect against all elements that are the result of some hardware-based rendering process. Similar to canvas fingerprinting, the processed signals resulting from AudioContext fingerprinting [12] will present differences due to the software and hardware stack of the device. Recently, the developer team

of AmIUnique released its second version. They now collect *WebGL Data*, this is a specific picture rendered with HTML5 Canvas and the WebGL API. Despite we do not know of any study that has conducted an analysis of the diversity of attributes collected through hardware-based rendering processes on virtualization, our observations indicate that if instead of implementing our strategy locally, on the user's machine, this is implemented in the cloud or on a server shared by several users, all these problems would be solved, since all users in principle would share the same hardware.

5.2 Threat to validity

There are two main elements could limit the scope of or strategy: the size of the sample and the components that make up fingerprints. The defense is based on a prior process of browser fingerprint collection. If the number of fingerprints collected is not large enough, it may introduce a certain level of bias, as it is not representative for most Internet devices. We certainly do not know the information that servers collect. We only analyze and protected against a very limited number of components. If we look at the Google Privacy Terms: "The information we collect includes unique identifiers, browser type and settings, device type and settings, operating system, (...). We also collect information about the interaction of your apps, browsers, and devices with our services, including IP address, crash reports, system activity, and the date, time, and referrer URL of your request." Google informs that information is being collected, but does not specify what kind of information in particular.

The clustering process entails some uncertainty. The success of the clustering algorithm depends largely on the distance metric used. The metric might not properly reflect the ability of determining how close two fingerprints are. Local distances and the weights associated with each attribute have a great influence in this aspect. We computed weights based on observations made by [14], which is the most recent analysis on browser fingerprint diversity. By the other hand, the selection of the clustering algorithm itself was not deterministic.

5.3 Web technologies

The effectiveness of browser fingerprinting against possible technical evolutions is a topic that has already been discussed by several studies. Laperdrix et al. [18] and Gómez-Boix et al. [14] recreated some possible scenarios in which web technologies evolve. They demonstrated that the end of browser plugins reduces significantly the effectiveness of browser fingerprinting at uniquely identifying users. They also showed that by removing all features collected through JavaScript, fingerprint uniqueness drastically drops.

We argue that the best defense against browser fingerprint is that one that comes from the web browser itself. Due to browser fingerprinting depends on current web technologies, browsers from their own code can limit the effectiveness of this tracking technique. Evidence of this is that several browsers already include protection to defend against it. Pale Moon [1], Brave [3] and the Tor Browser [4] were the very first ones to add barriers against techniques like Canvas or WebGL fingerprinting.

As part of the Tor Uplift program [2], Mozilla have included some defense mechanisms against this tracking technique [22]. Mozilla affirms that fingerprinting violates Firefox's anti-tracking policy. Mozilla scheduled to add an anti-fingerprinting technique called "letterboxing" to Firefox with the release of version 67. The browser window can have any shape and size, but the page content is only displayed at certain preset dimensions and the rest is filled with a gray space. This technique was originally developed for the Tor Browser in 2015. Mozilla announced that in the (alpha) Firefox Nightly 68 and Firefox Beta 67 versions of the browser, users can proactively block any websites that are known to employ fingerprinting techniques [30]. At WWDC 2018, Apple announced that it plans to build anti-fingerprint tracking into its Safari browser. Apple introduced in Safari a fingerprinting defense mechanism that basically attempts to share the minimum amount of information that the web site needs to load properly [7].

Modern browsers are getting equipped with mitigation techniques that require a lot of development to integrate and maintain. Even so, if the defense comes embedded in the web browser it will not generate any inconsistency or behavior that stands out, since all devices adopt the same characteristic.

6 CONCLUSION

In this paper, we proposed an approach that by means of the identification of devices prone to share the same fingerprint, provides users with a common, shared configuration that exhibits an indistinguishable browser fingerprint. Through clustering algorithms based on distance functions our approach minimizes the identifiability of groups of devices while minimizes the number of changes in the configuration of web browsers. We propose a solution built on virtualization and modular architectures at the level of operating system and web browser. This new approach allows users to show real fingerprints that are more likely to be found in the wild. We also propose a novel measure to evaluate the identifiability of a set of devices, as well as a measure which evaluates the cost of adopting a new browser configuration from the configurations exhibited by a group of devices.

We designed a tool that through virtualization assembles a given web browser configuration that is common for all users who use it. Our empirical results showed that through virtualization it is possible to create a set of homogeneous web configurations shared by many users as possible. We intend to integrate all the tools that act independently into one and investigate whether it is possible to extend the scope of our tool to other operating systems. We also discussed that the latest changes on defense mechanisms embedded in web browsers may lead to a future in which web browsers do not leak as much information to web servers.

ACKNOWLEDGMENT

The authors would like to thank Franọis Taïani for providing insightful feedback while designing the strategy proposed. This work is partially supported by the CominLabs-PROFILE project and by the Wallenberg Autonomous Systems Program (WASP).

REFERENCES

[1] 2015. Pale Moon browser - Version 25.6.0 adds a canvas poisoning feature. https://www.palemoon.org/releasenotes.shtml.
[2] 2017. Fingerprinting protection in Firefox as part of the Tor Uplift Project – Mozilla Wiki. https://wiki.mozilla.org/Security/Fingerprinting.
[3] 2017. Fingerprinting Protection Mode – Brave browser. https://github.com/brave/browser-laptop/wiki/Fingerprinting-Protection-Mode.

[4] 2017. The Design and Implementation of the Tor Browser [DRAFT] "Cross-Origin Fingerprinting Unlinkability" – Tor Project Official website. https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkability.

[5] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 674–689. https://doi.org/10.1145/2660267.2660347

[6] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. ACM, New York, NY, USA, 1129–1140. https://doi.org/10.1145/2508859.2516674

[7] Apple. 2019. Safari. The best way to see the sites. https://www.apple.com/safari/

[8] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. 2016. Disguised Chromium Browser: Robust Browser, Flash and Canvas Fingerprinting Protection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society (WPES '16)*. ACM, New York, NY, USA, 37–46. https://doi.org/10.1145/2994620.2994621

[9] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. 2012. *User Tracking on the Web via Cross-Browser Fingerprinting*. Lecture Notes in Computer Science, Vol. 7161. Springer Berlin Heidelberg, Berlin, Heidelberg, 31–46. https://doi.org/10.1007/978-3-642-29615-4_4

[10] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *24nd Annual Network and Distributed System Security Symposium, NDSS*.

[11] Peter Eckersley. 2010. How Unique is Your Web Browser?. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies (PETS'10)*. Springer-Verlag, Berlin, Heidelberg, 1–18. http://dl.acm.org/citation.cfm?id=1881151.1881152

[12] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1388–1401. https://doi.org/10.1145/2976749.2978313

[13] Amin FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. 2015. FPGuard: Detection and Prevention of Browser Fingerprinting. In *Data and Applications Security and Privacy XXIX*. Lecture Notes in Computer Science, Vol. 9149. Springer International Publishing, 293–308. https://doi.org/10.1007/978-3-319-20810-7_21

[14] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *WWW2018 - TheWebConf 2018 : 27th International World Wide Web Conference*. Lyon, France. https://doi.org/10.1145/3178876.3186097

[15] Pierre Laperdrix. 2017. *Browser Fingerprinting: Exploring Device Diversity to Augment Authentification and Build Client-Side Countermeasures*. Ph.D. Dissertation. Rennes, INSA.

[16] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *9th International Symposium on Engineering Secure Software and Systems (ESSoS 2017)*. Bonn, Germany. https://hal.inria.fr/hal-01527580

[17] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2015. Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification. In *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*. Firenze, Italy. https://hal.inria.fr/hal-01121108

[18] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In *37th IEEE Symposium on Security and Privacy (S&P 2016)*. San Jose, United States. https://hal.inria.fr/hal-01285470

[19] JR Mayer. 2009. Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0 [Bachelor's degree thesis]. *Faculty of the Woodrow Wilson School of Public and International Affairs* (2009).

[20] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. 2017. Block me if you can: A large-scale study of tracker-blocking tools. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 319–333.

[21] Keaton Mowery and Hovav Shacham. 2012. Pixel Perfect: Fingerprinting Canvas in HTML5. In *Proceedings of W2SP 2012*, Matt Fredrikson (Ed.). IEEE Computer Society.

[22] Mozilla Developer Network and individual contributors. 2017. Firefox 52 for developers. https://developer.mozilla.org/en-US/Firefox/Releases/52

[23] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. PriVaricator: Deceiving Fingerprinters with Little White Lies. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 820–830. https://doi.org/10.1145/2736277.2741090

[24] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP '13)*. IEEE Computer Society, Washington, DC, USA, 541–555. https://doi.org/10.1109/SP.2013.43

[25] Takamichi Saito, Kazushi Takahashi, Koki Yasuda, Takayuki Ishikawa, Ko Takasu, Tomotaka Yamada, Naoki Takei, and Rio Hosoi. 2016. OS and Application Identification by Installed Fonts. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 684–689.

[26] Christof Torres, Hugo Jonker, and Sjouke Mauw. 2015. FP-Block: usable web privacy by controlling browser fingerprinting. In *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS 2015)*.

[27] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupé. 2019. Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1679–1696.

[28] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In *Proceedings of the 27th USENIX Security Symposium*.

[29] Tao Wang and Ian Goldberg. 2013. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES '13)*. ACM, New York, NY, USA, 201–212. https://doi.org/10.1145/2517840.2517851

[30] Mozilla Wiki. 2019. Firefox/Roadmap/Updates. https://wiki.mozilla.org/Firefox/Roadmap/Updates#2019-07-08

[31] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

APPENDIX A. DISTANCE METRIC FOR FINGERPRINT COMPARISON

Attributes **Cookies enabled**, **Use of local/session storage**, **Use of an ad blocker**, **Do Not Track**, **WebGL Vendor** and **WebGL Renderer** are represented by strings of characters and each attribute can take only one value at a time, each string has an unique meaning and there is no relation between values, these attributes are compared as nominal attributes. Equation 4 shows the local distance for nominal attributes comparison. **Platform** attribute is also included in this domain.

$$d(a_i, a_j) = \begin{cases} 0 & a_i = a_j \\ 1 & a_i \neq a_j \end{cases} \tag{4}$$

Attribute **Available of fonts** is represented by a set of elements. The distance between two sets is the proportion of elements that are only in one set, minus the proportion of elements that are in both (equation 5). The term is 0 if both sets are identical and 1 if they are completely different (same definition, but different formulation of the equation proposed by Laperdrix et al. [17]). The numerator of the equation corresponds to the *symmetric difference* or *disjunctive union*, being equivalent $(S_i \cup S_j) \setminus (S_i \cap S_j)$ and $S_i \triangle S_j$.

$$d(S_i, S_j) = \frac{|(S_i \cup S_j) \setminus (S_i \cap S_j)|}{|(S_i \cup S_j)|} \tag{5}$$

Attribute **Content Language** is represented by a list of elements, the order matters. Equation 6 shows a local distance for comparing lists of elements where the order matters. The distance between two lists of languages is the proportion of elements that are only in one set, minus the proportion of elements that are necessary to change of position, being the length of both lists the greatest number of possible movements. The functions $permut(L_1, L_2)$ computes the number of permutations that exist between L_1 and L_2. The element f is a value between 0 and 1, it means the weight of the change of positions, 0 means change of positions are not taken into account (list are represent as sets) and 1 means changes of positions have same importance than insertions and deletions.

$$d(L_i, L_j) = \frac{|L_i \triangle L_j| + permut(L_{si}, L_{sj}) * f}{|L_i| + |L_j|} \quad (6)$$

where $L_{si} = L_i \cap L_j$ keeping the order in which they appear according to L_i and $L_{sj} = L_i \cap L_j$ keeping the order in which they appear according to L_j

Attribute **List of plugins** is represented as sets of elements, each plugin is represented by its name, its version and a file name. Equation 7 shows the local distance for comparing list of plugins. The term $LP_{\neq name}$ represents the list of plugins with different names and the term $LP_{=name,\neq version}$ represents the list of plugins with different versions. Updating the version of a plugin is easier than install or uninstall a plugin. The term f is a value between 0 and 1, it weighs the version changing, 0 means that plugins with different versions are the same and 1 means than plugins with different versions are completely different, so changing the version of plugins has the same relevance than install or uninstall them.

$$distance(LP_i, LP_j) = \frac{|LP_{\neq name}| + |LP_{=name,\neq version}| * f}{|LP_i| + |LP_j|} \quad (7)$$

where $LP_{\neq name} = \{P \in (L_i \cup L_j) : \exists P_1 \in L_i \wedge \exists P_2 \in L_j, (P_{name} = P_{1name} \wedge P_{name} \neq P_{2name}) \vee (P_{name} = P_{2name} \wedge P_{name} \neq P_{1name})\}$
and $LP_{=name,\neq version} = \{P \in (L_i \cup L_j) : \exists P_1 \in L_i \wedge \exists P_2 \in L_j, P_{name} = P_{1name} \wedge P_{name} = P_{2name} \wedge P_{1version} \neq P_{2version}\}$

Attributes **Header Accept**, **Content encoding** and **List of HTTP headers** contain lists of elements, but we did not figure out the meaning of each value and mainly how to change their values, so they will be treated as nominal values, see Equation 4.