FP-STALKER: Tracking Browser Fingerprint Evolutions

Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy

FP-STALKER: Tracking Browser Fingerprint Evolutions

Antoine Vastel
Univ. Lille / Inria
antoine.vastel@univ-lille.fr

Pierre Laperdrix
INSA / Inria
pierre.laperdrix@inria.fr

Walter Rudametkin
Univ. Lille / Inria
walter.rudametkin@univ-lille.fr

Romain Rouvoy
Univ. Lille / Inria / IUF
romain.rouvoy@univ-lille.fr

***Abstract*—Browser fingerprinting has emerged as a technique to track users without their consent. Unlike cookies, fingerprinting is a stateless technique that does not store any information on devices, but instead exploits unique combinations of attributes handed over freely by browsers. The uniqueness of fingerprints allows them to be used for identification. However, browser fingerprints change over time and the effectiveness of tracking users over longer durations has not been properly addressed.**

In this paper, we show that browser fingerprints tend to change frequently—from every few hours to days—due to, for example, software updates or configuration changes. Yet, despite these frequent changes, we show that browser fingerprints can still be linked, thus enabling long-term tracking.

FP-STALKER is an approach to link browser fingerprint evolutions. It compares fingerprints to determine if they originate from the same browser. We created two variants of FP-STALKER, a rule-based variant that is faster, and a hybrid variant that exploits machine learning to boost accuracy. To evaluate FP-STALKER, we conduct an empirical study using $98,598$ fingerprints we collected from $1,905$ distinct browser instances. We compare our algorithm with the state of the art and show that, on average, we can track browsers for 54.48 days, and $26\,\%$ of browsers can be tracked for more than 100 days.

I. INTRODUCTION

Websites track their users for different reasons, including targeted advertising, content personalization, and security [2]. Traditionally, tracking consists in assigning unique identifiers to cookies. However, recent discussions and legislation have brought to light the privacy concerns these cookies imply; more people are sensitive to these issues. A study conducted by Microsoft in 2012 observed that they were unable to keep track of $32\,\%$ of their users using only cookies, as they were regularly deleted [26]. Cookie erasure is now common as many browser extensions and private modes automatically delete cookies at the end of browsing sessions.

In 2010, Eckersley introduced a tracking technique called *browser fingerprinting* that leverages the user's browser and system characteristics to generate a fingerprint associated to the browser [8]. He showed that $83.6\,\%$ of visitors to the PANOPTICLICK website[1] could be uniquely identified from a fingerprint composed of only 8 attributes. Further studies have focused on studying new attributes that increase browser fingerprint uniqueness [7], [10], [14], [18], [19], [20], while others have shown that websites use browser fingerprinting as a way to regenerate deleted cookies [1].

[1]https://panopticlick.eff.org

However, fingerprint uniqueness, by itself, is insufficient for tracking because fingerprints change. One needs to keep track of these evolutions to link them to previous fingerprints. Recent approaches exploit fingerprint uniqueness as a defense mechanism by adding randomness to break *uniqueness* [12], [13], [21], but they did not address *linkability*.

The goal of this paper is to *link* browser fingerprint evolutions and discover how long browsers can be tracked. More precisely, FP-STALKER detects if two fingerprints originate from the same *browser instance*, which refers to an installation of a browser on a device. Browser instances change over time, *e.g.* they are updated or configured differently, causing their fingerprints to evolve. We introduce two variants of FP-STALKER: a rule-based and an hybrid variant, which leverage rules and a random forest.

We evaluate our approach using $98,598$ browser fingerprints originating from $1,905$ browser instances, which we collected over two years. The fingerprints were collected using two browser extensions advertised on the AmIUnique website[2], one for Firefox[3] and the other for Chrome[4]. We compare both variants of FP-STALKER and an implementation of the algorithm proposed by Eckersley [8]. In our experiments, we evaluate FP-STALKER's ability to correctly link browser fingerprints originating from the same browser instance, as well as its ability to detect fingerprints that originate from unknown browser instances. Finally, we show that FP-STALKER can link, on average, fingerprints from a given browser instance for more than 51 days, which represents an improvement of 36 days compared to the closest algorithm from the literature.

In summary, this paper reports on four contributions:

1) We highlight the limits of browser fingerprint uniqueness for tracking purposes by showing that fingerprints change frequently ($50\,\%$ of browser instances changed their fingerprints in less than 5 days, $80\,\%$ in less than 10 days);
2) We propose two variant algorithms to link fingerprints from the same browser instance, and to detect when a fingerprint comes from an unknown browser instance;
3) We compare the accuracy of our algorithms with the state of the art, and we study how browser fingerprinting frequency impacts tracking duration;
4) Finally, we evaluate the execution times of our algorithms, and we discuss the impact of our findings.

[2]https://amiunique.org
[3]https://addons.mozilla.org/firefox/addon/amiunique/
[4]https://chrome.google.com/webstore/detail/amiunique/pigjfndpomdldkmoaiiigpbncemhjeca

TABLE I: An example of a browser fingerprint

Attribute	Source	Value Examples
`Accept`	HTTP header	text/html,application/xhtml+xml,application /xml;q=0.9,image/webp,*/*;q=0.8
`Connection`	HTTP header	close
`Encoding`	HTTP header	gzip, deflate, sdch, br
`Headers`	HTTP header	Connection Accept X-Real-IP DNT Cookie Accept-Language Accept-Encoding User-Agent Host
`Languages`	HTTP header	en-US,en;q=0.8,es;q=0.6
`User-agent`	HTTP header	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36
`Canvas`	JavaScript	Cwm fjordbank glyphs vext quiz, 😃
`Cookies`	JavaScript	yes
`Do not track`	JavaScript	yes
`Local storage`	JavaScript	no
`Platform`	JavaScript	MacIntel
`Plugins`	JavaScript	Plugin 0: Chrome PDF Viewer; ; mhiehjai. Plugin 1: Chrome PDF Viewer; Portable Document Format; internal-pdf-viewer. Plugin 2: Native Client; ; internal-nacl-plugin.
`Resolution`	JavaScript	2560x1440x24
`Timezone`	JavaScript	-180
`WebGL`	Javascript	NVIDIA GeForce GTX 750 Series; Microsoft .
`Fonts`	Flash	List of fonts installed on the device

The remainder of this paper is organized as follows. Section II gives an overview of the state of the art. Section III analyzes how browser fingerprints evolve over time. Section IV introduces Eckersley's algorithm as well as both variants of FP-STALKER. Section V reports on an empirical evaluation, a comparison to the state of the art, and a benchmark of our approach. Finally, we conclude in Section VI.

II. BACKGROUND & MOTIVATIONS

a) Browser fingerprinting: aims to identify web browsers without using stateful identifiers, like cookies [8]. A browser fingerprint is composed of a set of browser and system attributes. By executing a script in the browser, sensitive metadata can be revealed, including the browser's parameters, but also operating system and hardware details. As this technique is completely stateless, it remains hard to detect and block, as no information is stored on the client side. Individually, these attributes may not give away much information but, when combined, they often form a unique signature, hence the analogy with a fingerprint. Most of the attributes in a fingerprint are collected through JavaScript APIs and HTTP headers, but extra information can also be retrieved through plugins like Flash. Table I illustrates a browser fingerprint collected from a Chrome browser running on Windows 10.

b) Browser fingerprinting studies: have focused on uniquely identifying browsers. Mayer [17] was the first to point out, in 2009, that a browser's "*quirkiness*" that stems from its configuration and underlying operating system could be used for "*individual identification*". In 2010, the PANOPTICLICK study was the first large-scale demonstration of browser fingerprinting as an identification technique [8]. From about half a million fingerprints, Eckersley succeeded in uniquely identifying 83.6 % of browsers. Since then, many studies have been conducted on many different aspects of this tracking technique. As new features are included within web browsers to draw images, render 3D scenes or process sounds, new attributes have been discovered to strengthen the fingerprinting process [5], [7], [9], [10], [18], [19], [20]. Additionally, researchers have performed large crawls of the web that confirm a steady growth of browser fingerprinting [1], [2], [9], [22]. While most of these studies focused on desktops, others demonstrated they could successfully fingerprint mobile device browsers [11], [14]. Finally, a study we conducted in 2016 confirmed Eckersley's findings, but observed a notable shift in some attributes [14]. While the lists of plugins and fonts were the most revealing features in 2010, this has rapidly changed as the *Netscape Plugin Application Programming Interface* (NPAPI) has been deprecated in Chrome (September 2015) and Firefox (March 2017). Browser fingerprinting is continuously adapting to evolutions in browser technologies since highly discriminating attributes can change quickly.

c) Browser fingerprinting defenses: have been designed to counter fingerprint tracking. The largest part of a browser fingerprint is obtained from the JavaScript engine. However, the values of these attributes can be altered to mislead fingerprinting algorithms. Browser extensions, called *spoofers*, change browser-populated values, like the `User-agent` or the `Platform`, with pre-defined ones. The goal here is to expose values that are different from the real ones. However, Nikiforakis *et al.* showed that they may be harmful as they found that these extensions "*did not account for all possible ways of discovering the true identity of the browsers on which they are installed*" and they actually make a user "*more visible and more distinguishable from the rest of the users, who are using their browsers without modifications*" [22]. Torres *et al.* went a step further by providing the concept of *separation of web identities* with FP-BLOCK, where a browser fingerprint is generated for each encountered domain [24]. Every time a browser connects to the same domain, it will return the same fingerprint. However, it keeps presenting the same limitation as naive spoofers since the modified values are incomplete and can be incoherent. Laperdrix *et al.* explored the randomization of media elements, such as canvas and audio used in fingerprinting, to break fingerprint linkability [12]. They add a slight random noise to canvas and audio, that is not perceived by users, to defeat fingerprinting algorithms.

Finally, the TOR browser is arguably the best overall defense against fingerprinting. Their strategy is to have all users converge towards a normalized fingerprint. The TOR browser is a modified Firefox that integrates custom defenses [23]. In particular, they removed plugins, canvas image extraction is blocked by default, and well-known attributes have been modified to return the same information on all operating systems. They also defend against JavaScript font enumeration by bundling a set of default fonts with the browser. However, using TOR can degrade the user's experience (*e.g.*, due to latency) and can break some websites (*e.g.*, due to disabled features, websites that block the TOR network). Furthermore, the unique browser fingerprint remains limited, as changes to the browser's configuration, or even resizing the window, can make the browser fingerprint unique.

d) Browser fingerprint linkability: is only partially addressed by existing studies. Eckersley tried to identify returning users on the PANOPTICLICK website with a very simple heuristic based on string comparisons that made correct guesses 65 % of the time [8]. Although not related to browsers, the overall approach taken by Wu *et al.* to fingerprint Android smartphones from permissionless applications [25] is similar in nature to our work. They collected a 38-attribute fingerprint,

including the list of system packages, the storage capacity of the device and the current ringtone. Using a naive Bayes classifier, they were able to successfully link fingerprints from the same mobile device over time. However, the nature of the data in [25] strongly differs from the focus of this work. In particular, the attributes in a browser fingerprint are not composed of strong identifiers, like the current wallpaper, and the browser does not share personal information from other parts of the system as do applications on Android. For these reasons, the results are not comparable.

To the best of our knowledge, beyond the initial contribution by Eckersley, no other studies have looked into the use of advanced techniques to link browser fingerprints over time.

III. Browser Fingerprint Evolutions

This paper focuses on the *linkability of browser fingerprint evolutions over time*. Using fingerprinting as a long-term tracking technique requires not only obtaining unique browser fingerprints, but also linking fingerprints that originate from the same browser instance. Most of the literature has focused on studying or increasing fingerprint uniqueness [7], [8], [14]. While uniqueness is a critical property of fingerprints, it is also critical to understand fingerprint evolution to build an effective tracking technique. Our study provides more insights into browser fingerprint evolution in order to demonstrate the effectiveness of such a tracking technique.

a) Input dataset: The raw input dataset we collected contains $172,285$ fingerprints obtained from $7,965$ different browser instances. All browser fingerprints were obtained from AmIUnique extensions for Chrome and Firefox installed from July 2015 to early August 2017 by participants in this study. The extensions load a page in the background that fingerprints the browser. Compared to a fingerprinting website, the only additional information we collect is a unique identifier we generate per browser instance when the extension is installed. This serves to establish the ground truth. Moreover, we pre-process the raw dataset by applying the following rules:

1) We remove browser instances with less than 7 browser fingerprints. This is because to study the ability to track browsers, we need browser instances that have been fingerprinted multiple times.
2) We discard browser instances with inconsistent fingerprints due to the use of countermeasures that artificially alter the fingerprints. To know if a user installed such a countermeasure, we check if the browser or OS changes and we check that the attributes are consistent among themselves. Although countermeasures exist in the wild, they are used by a minority of users and, we argue, should be treated by a separate specialized anti-spoofing algorithm. We leave this task for future work.

After applying these rules, we obtain a final dataset of $98,598$ fingerprints from $1,905$ browser instances. All following graphs and statistics are based on this final dataset. Figure 1 presents the number of fingerprints and distinct browser instances per month over the two year period.

Most users heard of our extensions through posts published on popular tech websites, such as Reddit, Hackernews or Slashdot. Users install the extension to visualize the evolution

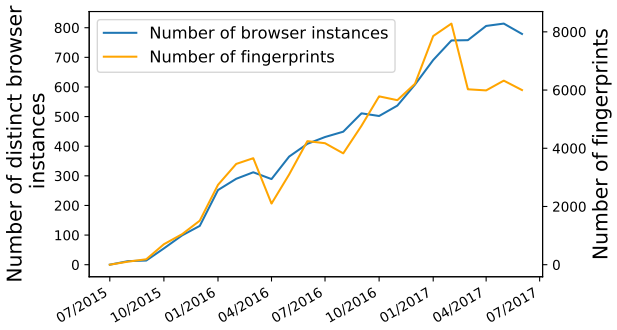


Fig. 1: Number of fingerprints and distinct browser instances per month

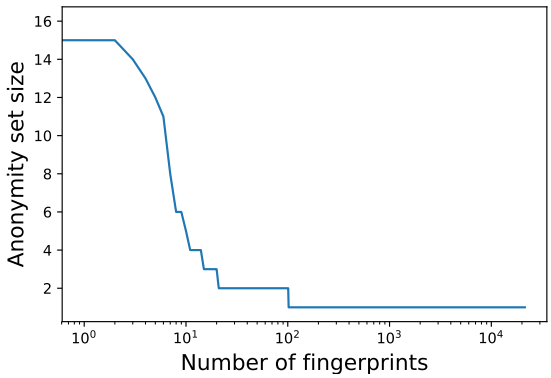


Fig. 2: Browser fingerprint anonymity set sizes

of their browser fingerprints over a long period of time, and also to help researchers understand browser fingerprinting in order to design better countermeasures. We explicitly state the purpose of the extension and the fact it collects their browser fingerprints. Moreover, we received an approval from the *Institutional Review Board* (IRB) of our research center for the collection as well as the storage of these browser fingerprints. As a ground truth, the extension generates a unique identifier per browser instance. The identifier is attached to all fingerprints, which are automatically sent every 4 hours. In this study, the browser fingerprints we consider are composed of the standard attributes described in Table I.

Figure 2 illustrates the anonymity set sizes against the number of participants involved in this study. The long tail reflects that $99\,\%$ of the browser fingerprints are unique among all the participants and belong to a single browser instance, while only 10 browser fingerprints are shared by more than 5 browser instances.

b) Evolution triggers: Browser fingerprints naturally evolve for several reasons. We identified the following categories of changes:

Automatic evolutions happen automatically and without direct user intervention. This is mostly due to automatic software upgrades, such as the upgrade of a browser or a plugin that may impact the `user agent` or the list of `plugins`;

Context-dependent evolutions being caused by changes in the user's context. Some attributes, such as `resolution` or `timezone`, are indirectly impacted by a contextual change , such as connecting a computer to an external screen or traveling to a different timezone; and

TABLE II: Durations the attributes remained constant for the median, the 90^{th} and the 95^{th} percentiles.

Attribute	Trigger	Percentile (days)		
		50th	90th	95th
Resolution	Context	Never	3.1	1.8
User agent	Automatic	39.7	13.0	8.4
Plugins	Automatic/User	44.1	12.2	8.7
Fonts	Automatic	Never	11.8	5.4
Headers	Automatic	308.0	34.1	14.9
Canvas	Automatic	290.0	35.3	17.2
Major browser version	Automatic	52.2	33.3	23.5
Timezone	Context	206.3	53.8	26.8
Renderer	Automatic	Never	81.2	30.3
Vendor	Automatic	Never	107.9	48.6
Language	User	Never	215.1	56.7
Dnt	User	Never	171.4	57.0
Encoding	Automatic	Never	106.1	60.5
Accept	Automatic	Never	163.8	109.5
Local storage	User	Never	Never	320.2
Platform	Automatic	Never	Never	Never
Cookies	User	Never	Never	Never

User-triggered evolutions that require an action from the user. They concern configuration-specific attributes, such as `cookies`, `do not track` or `local storage`.

To know how long attributes remain constant and if their stability depends on the browser instance, we compute the average time, per browser instance, that each attribute does not change. Table II presents the median, the 90^{th} and 95^{th} percentiles of the duration each attribute remains constant, on average, in browser instances. In particular, we observe that the `User agent` is rather unstable in most browser instances as its value is systematically impacted by software updates. In comparison, attributes such as `cookies`, `local storage` and `do not track` rarely change if ever. Moreover, we observe that attributes evolve at different rates depending on the browser instance. For example, `canvas` remains stable for 290 days in 50% of the browser instances, whereas it changes every 17.2 days for 10% of them. The same phenomena can be observed for the `screen resolution` where more than 50% of the browser instances never see a change, while 10% change every 3.1 days on average. More generally this points to some browser instances being quite stable, and thus, more trackable, while others aren't.

c) Evolution frequency: Another key indicator to observe is the elapsed time (E_t) before a change occurs in a browser fingerprint. Figure 3 depicts the cumulative distribution function of E_t for all fingerprints (blue), or averaged per browser instance (orange). After one day, at least one transition occurs in 45.2 % of the observed fingerprints. The 90^{th} percentile is observed after 13 days and the 95^{th} percentile after 17.3 days. This means the probability that at least one transition occurs in 13 days is 0.9 (blue). It is important to point out that changes occur more or less frequently depending on the browser instance (orange). While some browser instances change often (20% change in less than two days) others, on the contrary, are much more stable (23% have no changes after 10 days). In this context, keeping pace with the frequency of change is likely a challenge for browser fingerprint linking algorithms and, to the best of our knowledge, has not been explored in the state of the art.

d) Evolution rules: While it is difficult to anticipate browser fingerprint evolutions, we can observe how individual

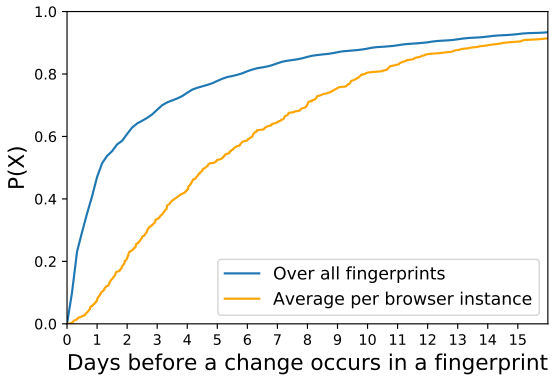


Fig. 3: CDF of the elapsed time before a fingerprint evolution for all the fingerprints, and averaged per browser instance.

attributes evolve. In particular, evolutions of the `User agent` attribute are often tied to browser upgrades, while evolutions of the `Plugins` attribute refers to the addition, deletion or upgrade of a plugin (upgrades change its version). Nevertheless, not all attribute changes can be explained in this manner, some values are difficult to anticipate. For example, the value of the `canvas` attribute is the result of an image rendered by the browser instance and depends on many different software and hardware layers. The same applies, although to a lesser extent, to screen `resolution`, which can take unexpected values depending on the connected screen. Based on these observations, the accuracy of linking browser fingerprint evolutions depends on the inference of such evolution rules. The following section introduces the evolution rules we first identified empirically, and then learned automatically, to achieve an efficient algorithm to track browser fingerprints over time.

IV. Linking Browser Fingerprints

FP-STALKER's goal is to determine if a browser fingerprint comes from a known browser instance—*i.e.*, it is an evolution—or if it should be considered as from a new browser instance. Because fingerprints change frequently, and for different reasons (see section III), a simple direct equality comparison is not enough to track browsers over long periods of time.

In FP-STALKER, we have implemented two variant algorithms with the purpose of linking browser fingerprints, as depicted in Figure 4. The first variant is a rule-based algorithm that uses a static ruleset, and the second variant is an hybrid algorithm that combines both rules and machine learning. We explain the details and the tradeoffs of both algorithms in this section. Our results show that the rule-based algorithm is faster but the hybrid algorithm is more precise while still maintaining acceptable execution times. We have also implemented a fully random forest-based algorithm, but the small increase in precision did not outweigh the large execution penalty, so we do not present it further in this paper.

A. Browser fingerprint linking

When collecting browser fingerprints, it is possible that a fingerprint comes from a previous visitor—*i.e.*, a known browser instance—or from a new visitor—*i.e.*, an unknown browser instance. The objective of fingerprint linking is to

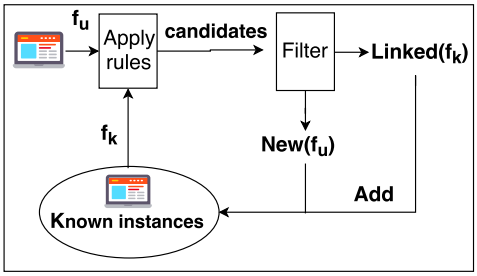
match fingerprints to their browser instance and follow the browser instance as long as possible by linking all of its fingerprint evolutions. In the case of a match, linked browser fingerprints are given the same identifier, which means the linking algorithm considers they originate from the same browser instance. If the browser fingerprint cannot be linked, the algorithm assigns a new identifier to the fingerprint.

More formally, given a set of known browser fingerprints F, each $f \in F$ has an identifier $f.id$ that links to the browser instance it belongs to. Given an unknown fingerprint $f_u \notin F$ for whom we ignore the real id, a linking algorithm returns the browser instance identifier $f_k.id$ of the fingerprint f_k that maximizes the probability that f_k and f_u belong to the same browser instance. This computation can be done either by applying rules, or by training an algorithm to predict this probability. If no known fingerprint can be found, it assigns a new id to f_u. For optimization purposes, we only hold and compare the last ν fingerprints of each browser instance b_i in F. The reason is because if we linked, for example, 3 browser fingerprints f_A, f_B and f_C to a browser instance b_i then, when trying to link an unknown fingerprint f_u, it is rarely useful to compare f_u to the oldest browser fingerprints of b_i. That is, newer fingerprints are more likely to produce a match, hence we avoid comparing old fingerprints in order to improve execution times. In our case we set the value of ν to 2.

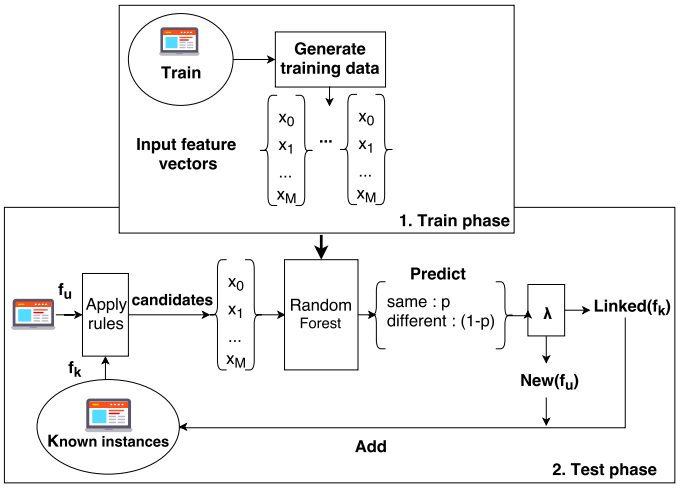
B. Rule-based Linking Algorithm

The first variant of FP-STALKER is a rule-based algorithm that uses static rules obtained from statistical analyses performed in section III. The algorithm relies on rules designed from attribute stability presented in Table II to determine if an unknown fingerprint f_u belongs to the same browser instance as a known fingerprint f_k. We also define rules based on constraints that we would not expect to be violated, such as, a browser's family should be constant (e.g., the same browser instance cannot be Firefox one moment and Chrome at a later time), the Operating System is constant, and the browser version is either constant or increases over time. The full list of rules are as follow:

1) The `OS`, `platform` and `browser family` must be identical for any given browser instance. Even if this may not always be true (*e.g.* when a user updates from Windows 8 to 10), we consider it reasonable for our algorithm to lose track of a browser when such a large change occurs since it is not frequent.
2) The `browser version` remains constant or increases over time. This would not be true in the case of a downgrade, but this is also, not a common event.
3) Due to the results from our statistical analyses, we have defined a set of attributes that must not differ between two fingerprints from the same browser instance. We consider that `local storage`, `Dnt`, `cookies` and `canvas` should be constant for any given browser instance. As observed in Table II, these attributes do not change often, if at all, for a given browser instance. In the case of `canvas`, even if it seldomly changes for most users (see Table II , the changes are unpredictable making them hard to model. Since `canvas` are quite unique among browser instances [14], and don't change too frequently, it is still interesting to consider that it must remain identical between two fingerprints of the same browser instance.
4) We impose a constraint on fonts: if both fingerprints have Flash activated—*i.e.* we have a list of fonts available—then the fonts of f_u must either be a subset or a superset of the fonts of f_k, but not a disjoint set. That means that between two fingerprints of a browser instance, it will allow deletions or additions of fonts, but not both.
5) We define a set of attributes that are allowed to change, but only within a certain similarity. That means that their values must have a similarity ratio > 0.75, as defined in the Python library function `difflib.SequenceMatcher().ratio`. These attributes are `user agent`, `vendor`, `renderer`, `plugins`, `language`, `accept`, `headers`. We allow at most two changes of this kind.
6) We also define a set of attributes that are allowed to change, no matter their value. This set is composed of `resolution`, `timezone` and `encoding`. However, we only allow one change at the same time among these three attributes.
7) Finally, the total number of changes from rules 5 and 6 must be less than 2.



(a) Rule-based variant of FP-STALKER. Uses a set of static rules to determine if fingerprints should be linked to the same browser instance or not.



(b) Hybrid variant of FP-STALKER. The training phase is used to learn the probability that two fingerprints belong to the same browser instance, and the testing phase uses the random forest-based algorithm to link fingerprints.

Fig. 4: FP-STALKER: Overview of both algorithm variants. The rule-based algorithm is simpler and faster but the hybrid algorithm leads to better fingerprint linking.

Algorithm 1 Rule-based matching algorithm

```
function FingerprintMatching(F, f_u)
    rules = {rule_1, ..., rule_6}
    candidates ← ∅
    exact ← ∅
    for f_k ∈ F do
        if VerifyRules(f_k, f_u, rules) then
            if nbDiff = 0 then
                exact ← exact ∪ ⟨f_k⟩
            else
                candidates ← candidates ∪ ⟨f_k⟩
            end if
        end if
    end for
    if |exact| > 0 and SameIds(exact) then
        return exact[0].id
    else if |candidates| > 0 and SameIds(candidates) then
        return candidates[0].id
    else
        return GenerateNewId()
    end if
end function
```

SameIds is a function that, given a list of candidates, returns true if all of them share the same id, else false.

The order in which rules are applied is important for performance purposes: we ordered them from the most to least discriminating. The first rules discard many candidates, reducing the total number of comparisons. In order to link f_u to a fingerprint f_k, we apply the rules to each known fingerprint taken from F. As soon as a rule is not matched, the known fingerprint is discarded and we move onto the next. If a fingerprint matches all the rules, then it is added to a list of potential candidates, $candidates$. Moreover, in case fingerprints f_k and f_u are identical, we add it to the list of exact matching candidates, $exact$. Once the rule verification process is completed, we look at the two lists of candidates. If $exact$ is not empty, we check if there is only one candidate or if all the candidates come from the same browser instance. If it is the case, then we link f_u with this browser instance, otherwise we assign a new id to f_u. In case no exact candidate is found, we look at $candidates$ and apply the same technique as for $exact$. We summarize the rule-based approach in Algorithm 1.

On a side-note, we established the rules using a simple univariate statistical analysis to study attribute stability (see Table II), as well as some objective (*e.g.,* rule 1) and other subjective (*e.g.,* rule 4) decisions. Due to the difficulty in making complex yet effective rules, the next subsection presents the use of machine learning to craft a more effective algorithm.

C. Hybrid Linking Algorithm

The second variant of FP-STALKER mixes the rule-based algorithm with machine learning to produce a hybrid algorithm. It reuses the first three rules of the previous algorithm, since we consider them as constraints that should not be violated between two fingerprints of a same browser instance. However, for the last four rules, the situation is more fuzzy. Indeed, it is not as clear when to allow attributes to be different, how many of them can be different, and with what dissimilarity. Instead of manually crafting rules for each of these attributes, we propose to use machine learning to discover them. The interest of combining both rules and machine learning approaches is that rules are faster than machine learning, but machine learning tends to be more precise. Thus, by applying the rules first, it helps keep only a subset of fingerprints on which to apply the machine learning algorithm.

1) Approach Description: The first step of this algorithm is to apply rules 1, 2 and 3 on f_u and all $f_k \in F$. We keep the subset of browser fingerprints f_{ksub} that verify these rules. If, during this process, we found any browser fingerprints that exactly match f_u, then we add them to $exact$. In case $exact$ is not empty and all of its candidates are from the same browser instance, we stop here and link f_u with the browser instance in $exact$. Otherwise, if there are multiple exact candidates but from different browser instances, then we assign a new browser id to f_u. In the case where the set of exact candidates is empty, we continue with a second step that leverages machine learning. In this step, for each fingerprint $f_k \in f_{ksub}$, we compute the probability that f_k and f_u come from the same browser instance using a random forest model. We keep a set of fingerprint candidates whose probability is greater than a λ threshold parameter. If the set of candidates is empty, we assign a new id to f_u. Otherwise, we keep the set of candidates with the highest and second highest probabilities, c_{h1} and c_{h2}. Then, we check if c_{h1} contains only one candidate or if all of the candidates come from the same browser instance. If it is not the case, we check that either the probability p_{h1} associated with candidates of c_{h1} is greater than the probability p_{h2} associated with candidates of $c_{h2} + diff$, or that c_{h2} and c_{h1} contains only candidates from the same browser instance. Algorithm 2 summarizes the hybrid approach.

2) Machine Learning: Computing the probability that two fingerprints f_u and f_k originate from the same browser instance can be modeled as a binary classification problem where the two classes to predict are `same browser instance` and `different browser instance`. We use the random forest algorithm [6] to solve this binary classification problem. A random forest is an ensemble learning method for classification that operates by constructing a multitude of decision trees at training time and outputting the class of the individual trees. In the case of FP-STALKER, each decision tree makes a prediction and votes if the two browser fingerprints come from the same browser instance. The result of the majority vote is chosen. Our main motivation to adopt a random forest instead of other classifiers is because it provides a good tradeoff between precision and the interpretation of the model. In particular, the notion of feature importance in random forests allows FP-STALKER to interpret the importance of each attribute in the decision process.

In summary, given two fingerprints, $f_u \notin F$ and $f_k \in F$, whose representation is reduced to a single feature vector of M features $X = \langle x_1, x_2, ..., x_M \rangle$, where the feature x_n is the comparison of the attribute n for both fingerprints (the process of transforming two fingerprints into a feature vector is presented after). Our random forest model computes the probability $P(f_u.id = f_k.id \mid (x_1, x_2, ..., x_M))$ that f_u and f_k belong to the same browser instance.

a) Input Feature Vector: To solve the binary classification problem, we provide an input vector $X = \langle x_1, x_2, ..., x_M \rangle$ of M features to the random forest classifier. The features are mostly pairwise comparisons between the values of the attributes of both fingerprints (*e.g.*, `Canvas`, `User agent`).

Algorithm 2 Hybrid matching algorithm

function FINGERPRINTMATCHING(F, f_u, λ)
  $rules = \{rule_1, rule_2, rule_3\}$
  $exact \leftarrow \emptyset$
  $F_{ksub} \leftarrow \emptyset$
  for $f_k \in F$ **do**
    if VERIFYRULES($f_k, f_u, rules$) **then**
      if $nbDiff = 0$ **then**
        $exact \leftarrow exact \cup \langle f_k \rangle$
      else
        $F_{ksub} \leftarrow F_{ksub} \cup \langle f_k \rangle$
      end if
    end if
  end for
  if $|exact| > 0$ **then**
    if SAMEIDS($exact$) **then**
      return $exact[0].id$
    else
      return GENERATENEWID()
    end if
  end if
  $candidates \leftarrow \emptyset$
  for $f_k \in F_{ksub}$ **do**
    $\langle x_1, x_2, ..., x_M \rangle = \text{FEATUREVECTOR}(f_u, f_k)$
    $p \leftarrow P(f_u.id = f_k.id \mid \langle x_1, x_2, ..., x_M \rangle)$
    if $p \geq \lambda$ **then**
      $candidates \leftarrow candidates \cup \langle f_k, p \rangle$
    end if
  end for
  if $|candidates| > 0$ **then**
    $c_{h1}, p_{h1} \leftarrow$ GETCANDIDATESRANK($candidates, 1$)
    $c_{h2}, p_{h2} \leftarrow$ GETCANDIDATESRANK($candidates, 2$)
    if SAMEIDS(c_{h1}) **and** $p_{h1} > p_{h2} + diff$ **then**
      return $candidates[0].id$
    end if
    if SAMEIDS($c_{h1} \cup c_{h2}$) **then**
      return $candidates[0].id$
    end if
  end if
  return GENERATENEWID()
end function

GETCANDIDATESRANK is a function that given a list of candidates and an rank i, returns a list of candidates with the ith greatest probability, and this probability.

Most of these features are binary values (`0` or `1`) corresponding to the equality or inequality of an attribute, or similarity ratios between these attributes. We also include a `number of changes` feature that corresponds to the total number of different attributes between f_u and f_k, as well as the time difference between the two fingerprints.

In order to choose which attributes constitute the feature vector we made a feature selection. Indeed, having too many features does not necessarily ensure better results. It may lead to *overfitting—i.e.*, our algorithm correctly fits our training data, but does not correctly predict on the test set. Moreover, having too many features also has a negative impact on performance. For the feature selection, we started with a model using all of the attributes in a fingerprint. Then, we looked at feature importance, as defined by [15], to determine the most discriminating features. In our case, feature importance is a combination of uniqueness, stability, and predictability (the possibility to anticipate how an attribute might evolve over time). We removed all the components of our feature vector that had a negligible impact (feature importance < 0.002). Finally, we obtained a feature vector composed of the attributes presented in Table III. We see that the most important feature is the number of differences between two fingerprints, and the second most discriminating attribute is the list of languages. Although this may seem surprising since the list of languages does not have high entropy, it does remain stable over time, as shown in Table II, which means that if two fingerprints have different languages, this often means that they do not belong to the same browser instance. In comparison, screen resolution also has low entropy but it changes more often than the list of languages, leading to low feature importance. This is mostly caused by the fact that since screen resolution changes frequently, having two fingerprints with a different resolution doesn't add a lot of information to determine whether or not they are from the same browser instance. Finally, we see a high drop in feature importance after rank 5 (from 0.083 to 0.010), which means that most of the information required for the classification is contained in the first five features.

TABLE III: Feature importances of the random forest model calculated from the fingerprint train set.

Rank	Feature	Importance
1	`Number of changes`	0.350
2	`Languages HTTP`	0.270
3	`User agent HTTP`	0.180
4	`Canvas`	0.090
5	`Time difference`	0.083
6	`Plugins`	0.010
7	`Fonts`	0.008
8	`Renderer`	0.004
9	`Resolution`	0.003

b) Training Random Forests: This phase trains the random forest classifier to estimate the probability that two fingerprints belong to the same browser instance. To do so, we split the input dataset introduced in Section III chronologically into two sets: a training set and a test set. The training set is composed of the first $40\,\%$ of fingerprints in our input dataset, and the test set of the last $60\,\%$. The random forest detects fingerprint evolutions by computing the evolutions between fingerprints as feature vectors. During the training phase, it needs to learn about correct evolutions by computing relevant feature vectors from the training set. Algorithm 3 describes this training phase, which is split into two steps.

Algorithm 3 Compute input feature vectors for training

function BUILDTRAININGVECTORS(ID,F,δ,ν)
  $T \leftarrow \emptyset$
  for $id \in ID$ **do** ▷ Step 1
    $F_{id} \leftarrow$ BROWSERFINGERPRINTS(id, F)
    for $f_t \in F_{id}$ **do**
      $T \leftarrow T \cup \text{FEATUREVECTOR}(f_t, f_{t-1})$
    end for
  end for
  for $f \in F$ **do** ▷ Step 2
    $f_r \leftarrow$ RANDOM(F)
    if $f.id \neq f_r.id$ **then**
      $T \leftarrow T \cup \text{FEATUREVECTOR}(f, f_r)$
    end if
  end for
  return T
end function

In `Step 1`, for every browser instance (id) of the training set, we compare each of its fingerprints ($f_t \in$ BROWSERFINGERPRINTS(id, F)) present in the training set (F) with the previous one (f_{t-1}). By doing so, FP-STALKER captures the atomic evolutions that occur between two consecutive fingerprints from the same browser instance. We apply BUILDTRAININGVECTORS() for different collect frequencies (time difference between t and $t-1$) to teach our model to link fingerprints even when they are not equally spaced in time.

While `Step 1` teaches the random forest to identify fingerprints that belong to the same browser instance, it is also necessary to identify when they do not. `Step 2` compares fingerprints from different browser instances. Since the number of fingerprints from different browser instances is much larger than the number of fingerprints from the same browser instance, we limit the number of comparisons to one for each fingerprint. This technique is called *undersampling* [16] and it reduces overfitting by adjusting the ratio of input data labeled as *true—i.e.*, 2 fingerprints belong to the same browser instance—against the number of data labeled as *false—i.e.*, 2 fingerprints are from different browser instances. Otherwise, the algorithm would tend to simply predict *false*.

c) Random forest hyperparameters.: Concerning the *number of trees* of the random forest, there is a tradeoff between precision and execution time. Adding trees does obtain better results but follows the law of diminishing returns and increases training and prediction times. Our goal is to balance precision and execution time. The *number of features* plays a role during the tree induction process. At each split, N_f features are randomly selected, among which the best split is chosen [4]. Usually, its default value is set to the square root of the length of the feature vector. The *diff* parameter enables the classifier to avoid selecting browser instances with very similar probabilities as the origin of the fingerprint; we would rather create a new browser instance than choose the wrong one. It is not directly related to random forest hyperparameters but rather to the specificities of our approach. In order to optimize the hyperparameters *number of trees* and *number of features*, as well as the *diff* parameter, we define several possible values for each and run a grid search to optimize the accuracy. This results in setting the hyperparameters to 10 trees and 3 features, and the *diff* value to 0.10.

After training our random forest classifier, we obtain a forest of decision trees that predict the probability that two fingerprints belong to the same browser instance. Figure 5 illustrates the first three levels of one of the decision trees. These levels rely on the `languages`, the `number of changes` and the `user agent` to take a decision. If an attribute has a value below its threshold, the decision path goes to the left child node, otherwise it goes to the right child node. The process is repeated until we reach a leaf of the tree. The prediction corresponds to the class (same/different browser instance) that has the most instances over all the leaf nodes.

d) Lambda threshold parameter: For each browser fingerprint in the test set, we compare it with its previous browser fingerprint and with another random fingerprint from a different browser, and compute the probability that it belongs to the same browser instance using our random forest classifier with the parameters determined previously. Using these probabilities and the true labels, we choose the λ value that minimizes the false positive rate, while maximizing the true positive rate. However, this configuration parameter depends on the targeted application of browser fingerprinting. For instance, if browser fingerprinting is used as a second-tier security mechanism (*e.g.*, to verify the user is connecting from a known browser instance), we set λ to a high value. This makes the algorithm more conservative, reducing the risk of linking a fingerprint to an incorrect browser instance, but it also increases false negatives and results in a reduction of the duration the algorithm can effectively track a browser. On the opposite end, a low λ value will increase the false positive rate, in this case meaning it tends to link browser fingerprints together even though they present differences. Such a use case might be acceptable for constructing ad profiles, because larger profiles are arguably more useful even if sometimes contaminated with someone else's information. By applying this approach, we obtained a λ threshold equal to 0.994.

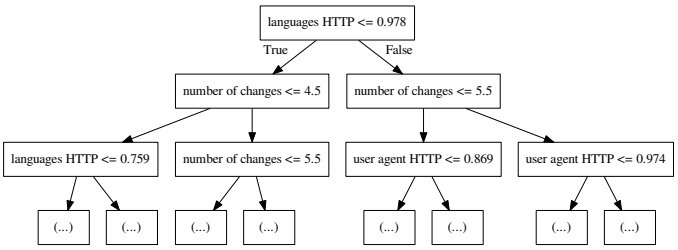


Fig. 5: First 3 levels of a single tree classifier from our forest.

V. EMPIRICAL EVALUATION OF FP-STALKER

This section assesses FP-STALKER's capacity to *i)* correctly link fingerprints from the same browser instance, and to *ii)* correctly predict when a fingerprint belongs to a browser instance that has never been seen before. We show that both variants of FP-STALKER are effective in linking fingerprints and in distinguishing fingerprints from new browser instances. However, the rule-based variant is faster while the hybrid variant is more precise. Finally, we discuss the impact of the collect frequency on fingerprinting effectiveness, and we evaluate the execution times of both variants of FP-STALKER.

Figure 6 illustrates the linking and evaluation process. Our database contains perfect tracking chains because of the unique identifiers our extensions use to identify browser instances. From there, we sample the database using different collection frequencies and generate a test set that removes the identifiers, resulting in a mix of fingerprints from different browsers. The resulting test set is then run through FP-STALKER to reconstruct the best browser instance chains as possible.

A. Key Performance Metrics

To evaluate the performance of our algorithms and measure how vulnerable users are to browser fingerprint tracking, we consider several metrics that represent the capacity to keep track of browser instances over time and to detect new browser instances. This section presents these evaluation metrics, as well as the related vocabulary. Figure 6 illustrates the different metrics with a scenario.

A *tracking chain* is a list of fingerprints that have been linked—*i.e.*, fingerprints for which the linking algorithm assigned the same identifier. A chain may be composed of one or

more fingerprints. In case of a perfect linking algorithm, each browser instance would have a unique tracking chain—*i.e.*, all of its fingerprints are grouped together and are not mixed with fingerprints from any other browser instances. However, in reality, fingerprinting is a statistical attack and mistakes may occur during the linking process, which means that:

1) Fingerprints from different browser instances may be included in the same tracking chain.
2) Fingerprints from a given browser instance may be split into different tracking chains.

The lower part of Figure 6 shows examples of these mistakes. *Chain 1* has an incorrect fingerprint `fpB1` from `Browser B`, and `chain 3` and `chain 4` contain fingerprints from `browser C` that have not correctly been linked—*i.e.*, `fpC3` and `fpC4` were not linked leading to a split).

We present the *tracking duration* metric to evaluate the capacity of an algorithm to track browser instances over time. We define *tracking duration* as the period of time a linking algorithm matches the fingerprints of a browser instance within a single tracking chain. More specifically, the tracking duration for a browser $\texttt{b}_i$ in a chain $\texttt{chain}_k$ is defined as $CollectFrequency \times (\#b_i \in chain_k - 1)$. We subtract one because we consider a browser instance to have been tracked, by definition, from the second linked fingerprint onwards.

The *average tracking duration* for a browser instance b_i is the arithmetic mean of its tracking duration across all the tracking chains the instance is present in. For example, in Figure 6, the tracking duration of `browser B` in `chain 1` is $0 \times CollectFrequency$, and the tracking duration in `chain 2` is $1 \times CollectFrequency$, thus the average tracking duration is $0.5 \times CollectFrequency$. In the same manner, the *average tracking duration* of *browser C* is $1.5 \times CollectFrequency$.

The *maximum tracking duration* for a browser instance b_i is defined as the maximum tracking duration across all of the tracking chains the browser instance is present in. In the case of `browser C`, the maximum tracking duration occurred in `chain 3` and is equal to $2 \times CollectFrequency$.

The *Number of assigned ids* represents the number of different identifiers that have been assigned to a browser instance by the linking algorithm. It can be seen as the number of tracking chains in which a browser instance is present. For each browser instance, a perfect linking algorithm would group all of the browser's fingerprints into a single chain. Hence, each browser instance would have a *number of assigned ids* of 1. Figure 6 shows an imperfect case where `browser C` has been assigned 2 different ids (`chain 3` and `chain 4`).

The *ownership ratio* reflects the capacity of an algorithm to not link fingerprints from different browser instances. The `owner` of a tracking chain $\texttt{chain}_k$ is defined as the browser instance $\texttt{b}_i$ that has the most fingerprints in the chain. Thus, we define *ownership ratio* as the number of fingerprints that belong to the `owner` of the chain divided by the length of the chain. For example, in `chain 1`, `browser A` owns the chain with an `ownership ratio` of $\frac{4}{5}$ because it has 4 out of 5 of the fingerprints. In practice, an *ownership ratio* close to 1 means that a tracking profile is not polluted with information from different browser instances.

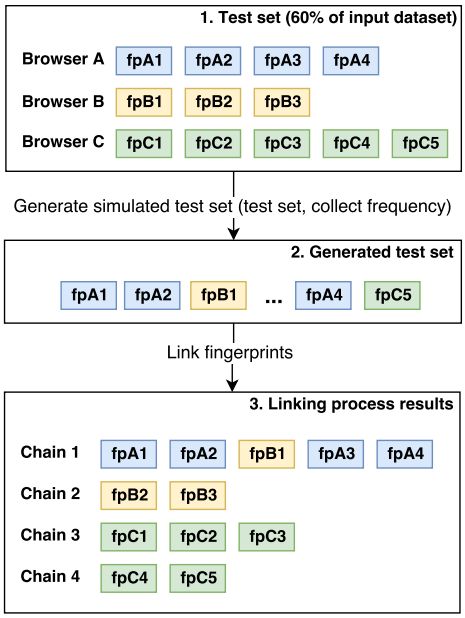


Fig. 6: Overview of our evaluation process that allows testing the algorithms using different simulated collection frequencies.

Algorithm 4 Eckersley fingerprint matching algorithm [8]

$ALLOWED = \{cookies, resolution, timezone, local\}$
function FINGERPRINTMATCHING(F, f_u)
  $candidates \leftarrow \emptyset$
  for $f_k \in F$ **do**
    $changes \leftarrow$ DIFF(f_u, f_k)
    if $|changes| = 1$ **then**
      $candidates \leftarrow candidates \cup \langle f_k, changes \rangle$
    end if
  end for
  if $|candidates| = 1$ **then**
    $\langle f_k, a \rangle \leftarrow candidates[0]$
    if $a \in ALLOWED$ **then**
      return f_k
    else if MATCHRATIO($f_u(a), f_k(a)$) > 0.85 **then**
      return f_k
    else
      return $NULL$
    end if
  end if
end function

MATCHRATIO refers to the Python standard library function `difflib.SequenceMatcher().ratio()` for estimating the similarity of strings.

B. Comparison with Panopticlick's linking algorithm

We compare FP-STALKER to the algorithm proposed by Eckersley [8] in the context of the PANOPTICLICK project. To the best of our knowledge, there are no other algorithms to compare to. Although Eckersley's algorithm has been characterized as "naive" by its author, we use it as a baseline to compare our approach. The PANOPTICLICK algorithm is summarized in Algorithm 4. It uses the following 8 attributes: `User agent`, `accept`, `cookies enabled`, `screen resolution`, `timezone`, `plugins`, `fonts` and `local storage`. Given an unknown fingerprint f_u, PANOPTICLICK tries to match it to a previous fingerprint of the same browser instance if a sufficiently similar one exists—

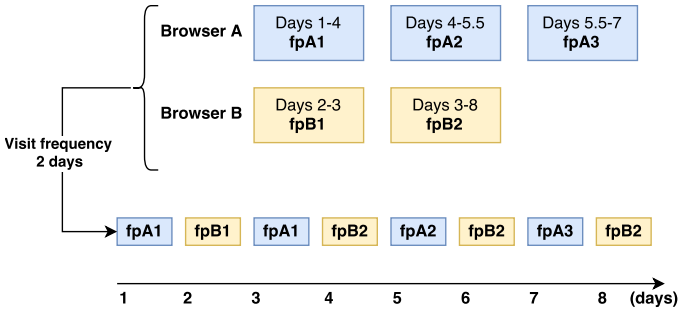


Fig. 7: Example of the process to generate a simulated test set. The dataset contains fingerprints collected from browser's `A` and `B`, which we sample at a $collect\_frequency$ of 2 days to obtain a dataset that allows us to test the impact of $collect\_frequency$ on fingerprint tracking.

i.e., no more than one attribute changed. Otherwise, if it found no similar fingerprints, or too many similar fingerprints that belong to different browser instances, it assigns a new id. Moreover, although at most one change is allowed, this change can only occur among the following attributes: `cookies`, `resolution`, `timezone` and `local storage`.

C. Dataset generation using fingerprint collect frequency

To evaluate the effectiveness of FP-STALKER we start from our test set of $59,159$ fingerprints collected from $1,395$ browser instances (60% of our input dataset, see Section IV-C2b). However, we do not directly use this set. Instead, by sampling the test set, we generate new datasets using a configurable collect frequency. Because our input dataset is fine-grained, it allows us to simulate the impact fingerprinting frequency has on tracking. The intuition being that if a browser is fingerprinted less often, it becomes harder to track.

To generate a dataset for a given collect frequency, we start from the test set of $59,159$ fingerprints, and, for each browser instance, we look at the collection date of its first fingerprint. Then, we iterate in time with a step of $collect\_frequency$ days and recover the browser instance's fingerprint at time $t + collect\_frequency$. It may be the same fingerprint as the previous collect or a new one. We do this until we reach the last fingerprint collected for that browser id. This allows us to record a sequence of fingerprints that correspond to the sequence a fingerprinter would obtain if the browser instance was fingerprinted at a frequency of $collect\_frequency$ days. The interest of sampling is that it is more realistic than using all of the fingerprints from our database since they are very fine-grained. Indeed, the extension is capable of catching even short-lived changes in the fingerprint (*e.g.*, connecting an external monitor), which is not always possible in the wild. Finally, it allows us to investigate how fingerprint collection frequency impacts browser tracking. Figure 7 provides an example of the process to generate a dataset with a $collect\_frequency$ of two days. Table IV presents, for each simulated collect frequency, the number of fingerprints in the generated test sets.

The browser fingerprints in a generated test set are ordered chronologically. At the beginning of our experiment, the set of known fingerprints (F) is empty. At each iteration, FP-STALKER tries to link an unknown fingerprint f_u with one of the fingerprints in F. If it can be linked to a fingerprint f_k, then FP-STALKER assigns the id $f_k.id$ to f_u, otherwise it assigns a new id. In both cases, f_u is added to F. The chronological order of the fingerprints implies that at time t, a browser fingerprint can only be linked with a former fingerprint collected at a time $t' < t$. This approach ensures a more realistic scenario, similar to online fingerprint tracking approaches, than if we allowed fingerprints from the past to be linked with fingerprints collected in the future.

TABLE IV: Number of fingerprints per generated test set after simulating different collect frequencies

Collect frequency (days)	Number of fingerprints
1	171,735
2	86,225
3	57,695
4	43,441
5	34,916
6	29,195
7	25,155
8	22,065
10	17,814
15	12,100
20	9,259

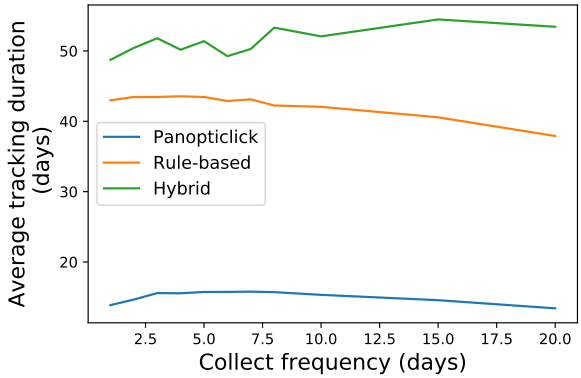


Fig. 8: Average *tracking duration* against simulated collect frequency for the three algorithms

D. Tracking duration

Figure 8 plots the average *tracking duration* against the collect frequency for the three algorithms. On average, browser instances from the test set were present for 109 days, which corresponds to the maximum value our linking algorithm could potentially achieve. We see that the hybrid variant of FP-STALKER is able to keep track of browser instances for a longer period of time than the two other algorithms. In the case where a browser gets fingerprinted every three days, FP-STALKER can track it for $51,8$ days, on average. More generally, the hybrid variant of FP-STALKER has an average tracking duration of about 9 days more than the rule-based variant and 15 days more than the Panopticlick algorithm.

Figure 9 presents the average *maximum tracking duration* against the collect frequency for the three algorithms. We see that the hybrid algorithm still outperforms the two other algorithms because the it constructs longer tracking chains with less mistakes. On average, the maximum average tracking duration for FP-STALKER's hybrid version is in the order of 74 days, meaning that at most users were generally tracked for this duration.

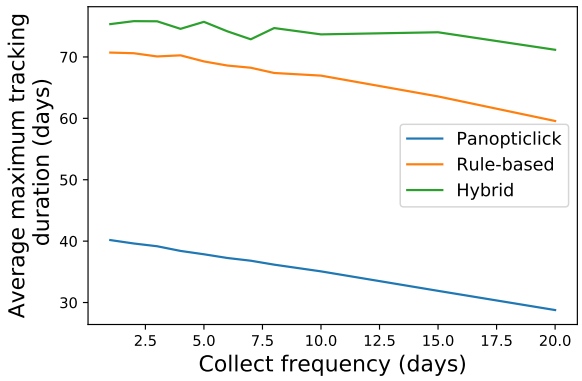


Fig. 9: Average maximum tracking duration against simulated collect frequency for the three algorithms. This shows averages of the longest tracking durations that were constructed.

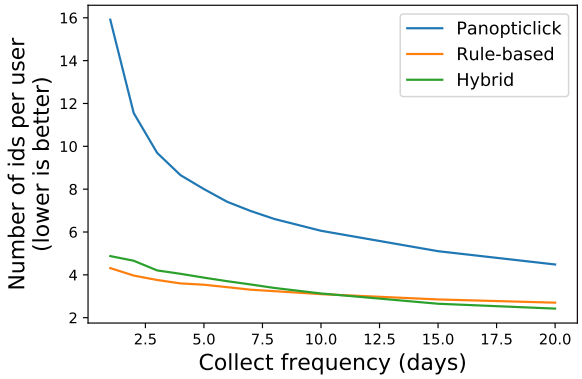


Fig. 10: Average number of assigned ids per browser instance against simulated collect frequency for the three algorithms (lower is better).

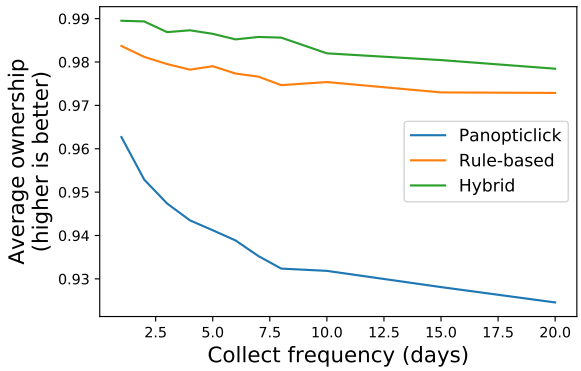


Fig. 11: Average ownership of tracking chains against simulated collect frequency for the three algorithms. A value of 1 means the tracking chain is constructed perfectly.

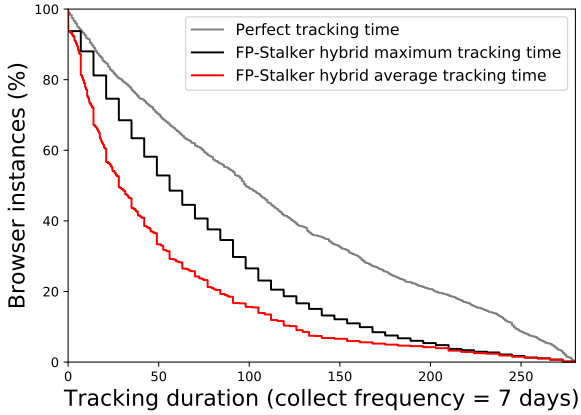


Fig. 12: CDF of average and maximum tracking duration for a collect frequency of 7 days (FP-STALKER hybrid variant only).

Figure 10 shows the *number of ids* they assigned, on average, for each browser instance. We see that PANOPTICLICK's algorithm often assigns new browser ids, which is caused by its conservative nature. Indeed, as soon as there is more than one change, or multiple candidates for linking, Panopticlick's algorithm assigns a new id to the unknown browser instance. However, we can observe that both FP-STALKER's hybrid and rule-based variants perform similarly.

Finally, Figure 11 presents the average *ownership* of tracking chains against the collect frequency for the three algorithms. We see that, despite its conservative nature, PANOPTICLICK's ownership is 0.94, which means that, on average, 6% of a tracking chain is constituted of fingerprints that do not belong to the browser instance that owns the chain—*i.e.*, it is contaminated with other fingerprints. The hybrid variant of FP-STALKER has an average *ownership* of 0.985, against 0.977 for the rule-based.

When it comes to linking browser fingerprints, FP-STALKER's hybrid variant is better, or as good as, the rule-based variant. The next paragraphs focus on a few more results we obtain with the hybrid algorithm. Figure 12 presents the cumulative distribution of the average and maximum tracking duration when $collect\_frequency$ equals 7 days for the hybrid variant. We observe that, on average, $15,5\%$ of the browser instances are tracked more than 100 days. When it comes to the the longest tracking chains, we observe that more than 26% of the browser instances have been tracked at least once for more than 100 days during the experiment. These numbers show how tracking may depend on the browser and its configuration. Indeed, while some browsers are never tracked for a long period of time, others may be tracked for multiple months. This is also due to the duration of presence of browser instances in our experiments. Few browser instances were present for the whole experiment, most for a few weeks, and at best we can track a browser instance only as long as it was present. The graph also shows the results of the perfect linking algorithm, which can also be interpreted as the distribution of duration of presence of browser instances in our test set.

The boxplot in Figure 13 depicts the *number of ids* generated by the hybrid algorithm for a collect frequency of 7 days. It shows that half of the browser instances have been assigned 2 identifiers, which means they have one mistake, and more than 90 % have less than 9 identifiers.

Finally, we also look at the distribution of the chains to see how often fingerprints from different browser instances are mixed together. For the FP-STALKER hybrid variant, more than 95% of the chains have an ownership superior to 0.8, and more than 90% have perfect ownership—*i.e.*, 1. This shows that a small percentage of browser instances become highly mixed in the chains, while the majority of browser instances are properly linked into clean and relatively long tracking chains.

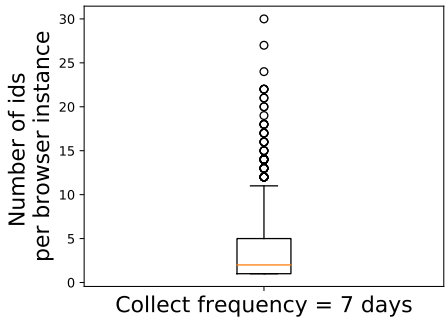


Fig. 13: Distribution of `number of ids` per browser for a collect frequency of 7 days (FP-STALKER hybrid variant only).

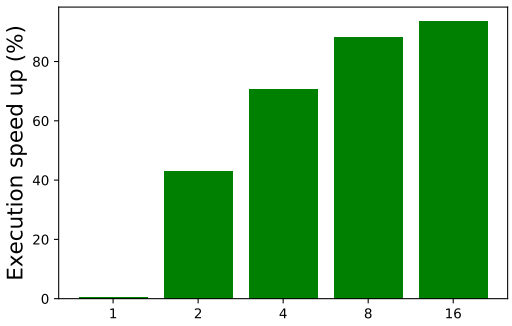


Fig. 14: Speedup of average execution time against number of processes for FP-STALKER's hybrid variant

E. Benchmark/Overhead

This section presents a benchmark that evaluates the performance of FP-STALKER's hybrid and rule-based variants. We start by providing more details about our implementation, then we explain the protocol used for this benchmark, demonstrate that our approach can scale, and we show how our two variants behave when the number of browser instances increases.

a) The implementations: of FP-STALKER used for this benchmark are developed in Python, and the implementation of the random forest comes from the Scikit-Learn library. In order to study the scalability of our approach, we parallelized the linking algorithm to run on multiple nodes. A master node is responsible for receiving linkability requests, then it sends the unknown fingerprint to match f_u to slave nodes that compare f_u with all of the f_k present on their process. Then, each slave node sends its set of candidates associated either with a probability in case of the hybrid algorithm, or the number of changes in case of the rule-based version. Finally, the master node takes the final decision according to the policy defined either by the rule-based or hybrid algorithm. After the decision is made, it sends a message to each node to announce whether or not they should keep f_u in their local memory. In the case of the benchmark, we do not implement an optimization for exact matching. Indeed, normally the master nodes should hold a list of the exact matches associated with their ids.

b) The experimental protocol: aims to study scalability. We evaluate our approach on a standard Azure cloud instance. We generate fake browser fingerprints to increase the test set size. Thus, this part does not evaluate the previous metrics, such as `tracking duration`, but only the execution times required to link synthetic browser fingerprints, as well as how well the approach scales across multiple processes.

The first step of the benchmark is to generate fake fingerprints from real ones. The generation process consists in taking a real fingerprint from our database and applying random changes to the canvas and the timezone attributes. We apply only two random changes so that generated fingerprints are unique, but they do not have too many differences which would reduce the number of comparisons. This point is important because our algorithms include heuristics related to the number of differences. Thus, by applying a small number of random changes, we do not discard all f_k fingerprints, making it the worst case scenario for testing execution times. Regarding the browser ids, we assign two generated fingerprints to each browser instance. It would not have been useful to generate more fingerprints per browser instance since we compare an unknown fingerprint only with the last 2 fingerprints of each browser instance. Then, the master node creates n slave processes and sends the generated fingerprints to them. The fingerprints are spread evenly over the processes.

Once the fingerprints are stored in the slave processes memory, we start our benchmark. We get 100 real fingerprints and try to link them with our generated fingerprints. For each fingerprint, we measure the execution time of the linking process. In this measurement, we measure:

1) The number of fingerprints and browser instances.
2) The number of processes spawned.

We execute our benchmark on a `Standard D16 v3` Azure instance with 16 virtual processors and 64 Gb of RAM, which has an associated cost of $576 USD per month. Figure 14 shows the execution time speedup in percentage against the number of processes for the hybrid approach. We see that that as the number of processes increases, we obtain a speedup in execution time. Going from 1 to 8 processes enables a speed up of more than 80 %. Figure 15 shows the execution time to link a fingerprint against the number of browser fingerprints for FP-STALKER's hybrid and rule-based variants, using 16 processes. Better `tracking duration` from the hybrid variant (see V-D) is obtained at the cost of execution speed. Indeed, for any given number of processes and browser instances, the rule-based variant links fingerprints about 5 times faster. That said, the results show that the hybrid variant links fingerprints relatively quickly.

However, the raw execution times should not be used directly. The algorithm was implemented in Python, whose primary focus is not performance. Moreover, although we scaled by adding processes, it is possible to scale further by splitting the linking process (*e.g.*, depending on the combination of OS and browser, send the fingerprint to more specialized nodes). In our current implementation, if an unknown fingerprint from a Chrome browser on Linux is trying to be matched, it will be compared to fingerprints from Firefox on Windows, causing us to wait even though they have no chance of being linked. By adopting a hierarchical structure where nodes or processes are split depending on their OS and browser, it is possible to increase the throughput of our approach.

Furthermore, the importance of the raw execution speeds depend highly on the use case. In the case where fingerprinting is used as a way to regenerate cookies (*e.g.*, for advertising), a fingerprint only needs to be linked when the cookie is missing

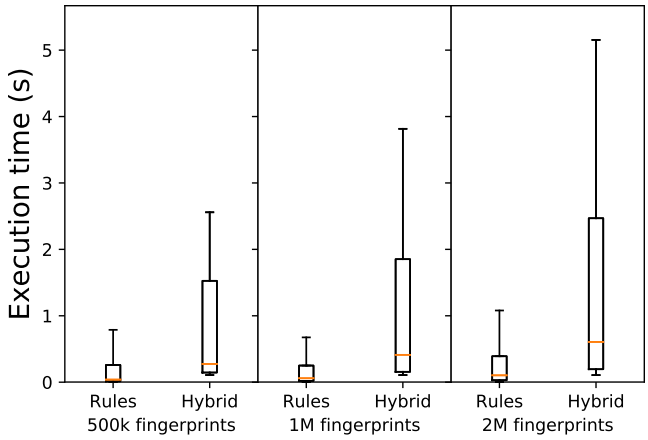


Fig. 15: Execution times for FP-STALKER hybrid and rule-based to link a fingerprint using 16 processes. Time is dependent on the size of the test set. The increased effectiveness of the hybrid variant comes at the cost slower of execution times.

or has been erased, a much less frequent event. Another use case is using browser fingerprinting as a way to enhance authentication [3]. In this case, one only needs to match the fingerprint of the browser attempting to sign-in with the previous fingerprints from the same user, drastically reducing the number of comparisons.

F. Threats to Validity

First, the results we report in this work depend on the representativity of our browser fingerprint dataset. We developed extensions for Chrome and Firefox, the two most popular web browsers, and distributed them through standard channels. This does provide long term data, and mitigates a possible bias if we had chosen a user population ourselves, but it is possible that the people interested in our extension are not a good representation of the average Web surfer.

Second, there is a reliability threat due to the difficulty in replicating the experiments. Unfortunately, this is inherent to scientific endeavors in the area of privacy: these works must analyze personal data (browser fingerprints in our case) and the data cannot be publicly shared. Yet, the code to split the data, generate input data, train the algorithm, as well as evaluate it, is publicly available online on GitHub[5].

Finally, a possible internal threat lies in our experimental framework. We did extensive testing of our machine learning algorithms, and checked classification results as thoroughly as possible. We paid attention to split the data and generate a scenario close to what would happen in a web application. However, as for any large scale experimental infrastructure, there are surely bugs in this software. We hope that they only change marginal quantitative things, and not the qualitative essence of our findings.

G. Discussion

This paper studies browser fingerprint linking in isolation, which is its worst-case scenario. In practice, browser fingerprinting is often combined with stateful tracking techniques (*e.g.*, cookies, Etags) to respawn stateful identifiers [1]. In such cases, fingerprint linking is performed much less frequently since most of the time a cookie is sufficient and inexpensive to track users. Our work shows that browser fingerprinting can provide an efficient solution to extend the lifespan of cookies, which are increasingly being deleted by privacy-aware users.

Browser vendors and users would do well to minimize the differences that are so easily exploited by fingerprinters. Our results show that some browser instances have highly trackable fingerprints, to the point that very infrequent fingerprinting is quite effective. In contrast, other browser instances appear to be untrackable using the attributes we collect. Vendors should work to minimize the attack surfaces exploited by fingerprinters, and users should avoid customizing their browsers in ways that make them expose unique and linkable fingerprints.

Depending on the objectives, browser fingerprint linking can be tuned to be more conservative and avoid false positives (e.g., for second-tier security purposes), or more permissive (e.g., ad tracking). Tuning could also be influenced by how effective other tracking techniques are. For example, it could be tuned very conservatively and simply serve to extend cookie tracking in cases where privacy-aware users, which are in our opinion more likely to have customized (i.e., unique and linkable) browser configurations, delete their cookies.

VI. CONCLUSION

In this paper, we investigated browser fingerprint evolution and proposed FP-STALKER as an approach to link fingerprint changes over time. We address the problem with two variants of FP-STALKER. The first one builds on a ruleset identified from an analysis of grounded programmer knowledge. The second variant combines the most discriminating rules by leveraging machine learning to sort out the more subtle ones.

We trained the FP-STALKER hybrid variant with a training set of fingerprints that we collected for 2 years through browser extensions installed by $1,905$ volunteers. By analyzing the feature importance of our random forest, we identified the `number of changes`, the `languages`, as well as the `user agent`, as the three most important features.

We ran FP-STALKER on our test set to assess its capacity to link fingerprints, as well as to detect new browser instances. Our experiments demonstrate that the hybrid variant can correctly link fingerprint evolutions from a given browser instance for 54.48 consecutive days on average, against $42,3$ days for the rule-based variant. When it comes to the maximum tracking duration, with the hybrid variant, more than 26% of the browsers can be tracked for more than 100 days.

Regarding the usability of FP-STALKER, we measure the average execution time to link an unknown fingerprint when the number of known fingerprints is growing. We show that both our rule-based and hybrid variants scale horizontally.

ACKNOWLEDGMENT

We would like to thank the users of the AmIUnique extensions, whose contributions were essential to this study. We also want to thank our shepherd, Davide Balzarotti, and the anonymous reviewers for their valuable comments and feedback. Finally, this work would not have been possible without our long-term collaboration with Benoit Baudry.

[5]https://github.com/Spirals-Team/FPStalker

REFERENCES

[1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild," *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, pp. 674–689, 2014.

[2] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "FPDetective," *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pp. 1129–1140, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2508859.2516674

[3] F. Alaca and P. C. V. Oorschot, "Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods," *Annual Computer Security Applications Conference (ASAC '32)*, 2016. [Online]. Available: http://people.scs.carleton.ca/~paulv/papers/acsac2016-device-fingerprinting.pdf

[4] S. Bernard, L. Heutte, and S. Adam, "Influence of hyperparameters on random forest accuracy," *Proc. Int. Workshop Multiple Classifier Syst.*, vol. 5519 LNCS, pp. 171–180, 2009.

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

[6] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[7] Y. Cao, "(Cross-) Browser Fingerprinting via OS and Hardware Level Features," *24nd Annual Network and Distributed System Security Symposium, NDSS'17*, no. March, 2017.

[8] P. Eckersley, "How unique is your web browser?" *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[9] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, no. 1, pp. 1388–1401, 2016.

[10] D. Fifield and S. Egelman, "Fingerprinting web users through font metrics," *Financial Cryptography and Data Security*, vol. 8975, pp. 107–124, 2015.

[11] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "On the Robustness of Mobile Device Fingerprinting," *Proceedings of the 31st Annual Computer Security Applications Conference*, pp. 191–200, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2818000.2818032

[12] P. Laperdrix, B. Baudry, and V. Mishra, "FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques," *Proceedings - 9th Int. Symposium on Engineering Secure Software and Systems, ESSoS 2017*, Jul. 2017. [Online]. Available: https://hal.inria.fr/hal-01527580

[13] P. Laperdrix, W. Rudametkin, and B. Baudry, "Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification," *Proceedings - 10th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*, pp. 98–108, 2015. [Online]. Available: https://hal.inria.fr/hal-01121108

[14] ——, "Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints," *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, pp. 878–894, 2016. [Online]. Available: https://hal.inria.fr/hal-01285470

[15] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," *Advances in Neural Information Processing Systems 26*, pp. 431–439, 2013. [Online]. Available: http://papers.nips.cc/paper/4928-understanding-variable-importances-in-forests-of-randomized-trees.pdf

[16] O. Loyola-González, M. García-Borroto, M. A. Medina-Pérez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and G. De Ita, *An Empirical Study of Oversampling and Undersampling Methods for LCMine an Emerging Pattern Based Classifier*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 264–273. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38989-4_27

[17] J. R. Mayer, "Internet Anonymity in the Age of Web 2.0," *A Senior Thesis presented to the Faculty of the Woodrow Wilson School of Public and International A airs in partial ful llment of the requirements for the degree of Bachelor of Arts.*, p. 103, 2009. [Online]. Available: https://jonathanmayer.org/papers_data/thesis09.pdf

[18] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, "Fingerprinting Information in JavaScript Implementations," *Web 2.0 Security & Privacy*, pp. 1–11, 2011. [Online]. Available: http://cseweb.ucsd.edu/~hovav/papers/mbys11.html

[19] K. Mowery and H. Shacham, "Pixel Perfect : Fingerprinting Canvas in HTML5," *Web 2.0 Security & Privacy 20 (W2SP)*, pp. 1–12, 2012.

[20] M. Mulazzani, P. Reschl, and M. Huber, "Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting," *Proceedings of W2SP*, 2013. [Online]. Available: http://www.sba-research.org/wp-content/uploads/publications/jsfingerprinting.pdf

[21] N. Nikiforakis, W. Joosen, and B. Livshits, "PriVaricator," *Proceedings of the 24th International Conference on World Wide Web - WWW '15*, pp. 820–830, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2736277.2741090

[22] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 541–555, 2013.

[23] M. Perry, E. Clark, and S. Murdoch, "The Design and Implementation of the Tor Browser," Tech. Rep., May 2015, https://www.torproject.org/projects/torbrowser/design.

[24] C. F. Torres, H. Jonker, and S. Mauw, "FP-block: Usable web privacy by controlling browser fingerprinting," *ESORICS, 2015*, vol. 9327, no. October, pp. 3–19, 2015.

[25] W. Wu, J. Wu, Y. Wang, Z. Ling, and M. Yang, "Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers," vol. 4, pp. 8073–8083, 2016.

[26] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, "Host Fingerprinting and Tracking on the Web: Privacy and Security Implications," *Network and Distributed System Security Symposium*, pp. 1–16, 2012.