Who Touched My Browser Fingerprint? A Large-scale Measurement Study and Classification of Fingerprint Dynamics

Song Li
Johns Hopkins University
lsong18@jhu.edu

Yinzhi Cao
Johns Hopkins University
yinzhi.cao@jhu.edu

ABSTRACT

Browser fingerprints are dynamic, evolving with feature values changed over time. Previous fingerprinting datasets are either small-scale with only thousands of browser instances or without considering fingerprint dynamics. Thus, it remains unclear how an evolution-aware fingerprinting tool behaves in a real-world setting, e.g., on a website with millions of browser instances, let alone how fingerprint dynamics implicate privacy and security.

In this paper, we perform the first, large-scale study of millions of fingerprints to analyze fingerprint dynamics in a real-world website. Our measurement study answers the question of how and why fingerprints change over time by classifying fingerprint dynamics into three categories based on their causes. We also observed several insights from our measurement, e.g., we show that state-of-the-art fingerprinting tool performs poorly in terms of F1-Score and matching speed in this real-world setting.

ACM Reference Format:
Song Li and Yinzhi Cao. 2020. Who Touched My Browser Fingerprint? A Large-scale Measurement Study and Classification of Fingerprint Dynamics. In *ACM Internet Measurement Conference (IMC '20), October 27–November 29, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3419394.3423614

1 INTRODUCTION

Browser fingerprinting, an alternative to browser cookies when being disabled or cleared, is that a website extracts a list of browser features at the client side and then constructs an identifier, called a fingerprint, based on these extracted features to identify or authenticate the browser. Browser fingerprinting is first studied by Eckerlsey [17] via his famous Panopticlick website [10] and now widely adopted by many tracking companies and real-world Alexa websites according to a recent study [18].

Prior works have measured browser fingerprints in the wild. On one hand, large-scale studies, such as Gómez-Boix et al. [20], have analyzed millions of browser fingerprints in the wild via collecting fingerprints on a real-world website. However, there are two major drawbacks. First, they only studied the effectiveness of fingerprints in differentiating and identifying browser instances but not how fingerprints evolve over time—which are called fingerprint dynamics in the paper. Second, prior works [20] adopt cookies as the ground truth—which rely on an assumption that people clear cookies but in a rare manner. This assumption is untrue as demonstrated in our study: 32% of browser instances clear cookies—one major cause is intelligent tracking prevention [6], which automatically deletes tracking cookies after a certain period.

On the other hand, there exists small-scale datasets with only thousands of fingerprints, such as Pugliese et al. [37] and the one used in FP-Stalker [40], an evolution-aware fingerprinting tool that links evolved fingerprints together. Those work usually adopt out-of-band identifiers, e.g., one provided via a browser extension, to recognize users. However, the requirement of out-of-band identifiers restrict the study scale: It is difficulty to let millions of users to install extensions for a measurement purpose.

In this paper, we perform the first large-scale measurement study of millions of fingerprints on a real-world website to analyze fingerprint dynamics, i.e., how browser fingerprints change over time and why they do so. Specifically, we implemented our version of fingerprinting tool and deployed it at a real-world European website visited regularly by its users, which collected a dataset with 7,246,618 fingerprints from 1,329,927 browser instances and 1,148,864 users. Our representation of browser instance is via a new type of identifier, called Browser ID, a combination of an anonymized username using hash values and some stable browser features. On one hand, Browser ID is much more stable as compared with cookies: The false positive of Browser ID of representing browser instances is estimated as 0.1% and the false negative rate as 0.3%; on the other hand, Browser ID can differentiate multiple devices of the same user: In our study, 14% users visit the deployment website using more than one device.

Next, we measure fingerprint dynamics by calculating the difference between two consecutive fingerprints of the same browser instance. The advantage of such diff operation over a simple fingerprint pair representation is that if two browser instances with different fingerprints (e.g., one instance with an additional font) get the same update (e.g., from Chrome 56 to 57), the delta information will also be the same. We produce a dataset of 960,853 dynamics—Our analysis of the dataset shows that all the dynamics can be classified into three major categories based on their causes: (i) browser or OS updates, (ii) user actions and (iii) system environment updates. Our further study of the dynamics dataset yields four insights:

- *Insight 1: Browser fingerprints, particularly the dynamics, reveal privacy- or security-related information.* The reason is that the

IMC '20, October 27–November 29, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8138-3/20/10…$15.00
https://doi.org/10.1145/3419394.3423614

cause of a piece of dynamics could contain privacy- or security-related information. For example, we find that a certain emoji update at a mobile Chrome browser can reveal the fact that a Samsung browser is co-installed with the Chrome browser because the Samsung update introduces a new emoji. Similarly, for another example, the font list and the changes of fonts in fingerprint dynamics can be used to infer whether Microsoft Office is installed or even updated.

- *Insight 2: The F1-score and matching speed of prior evolution-aware fingerprint work degrade significantly in a large-scale setting.* As stated, prior work, particularly FP-Stalker [40], is evaluated using a relatively small dataset with thousands of users and fingerprints to link evolved fingerprints. We find that the F1-score of rule-based FP-Stalker degrades from 86.1% to 75.9% for top ten candidates and the matching speed from around 100 ms to 1 second if the number of fingerprints increases from 100K to one million; the learning-based FP-Stalker cannot scale to a large-scale dataset with more than 300K fingerprints (the scalability issue of learning-based FP-Stalker is acknowledged in the original paper).
- *Insight 3: The dynamics of some browser features are correlated although the features themselves are not.* For example, we have observed that the sample rate of audio card in Chrome may change together with the GPU renderer. The reason is that although some features are not directly related, the causes behind the changes may be. Specifically, in the aforementioned example, Chrome adopts DirectX to manage audio card on certain Windows machines: An update of DirectX will influence both the GPU renderer and the audio sample rate.
- *Insight 4: The timing of some fingerprint dynamics are correlated with real-world events, such as the release of browser or OS updates.* We believe that such an insight might be used to improve the performance of existing works in linking fingerprints. For example, if Firefox updates to a new version with an added web font, a fingerprinting website can predict that all the fingerprints in the database with the old Firefox version may change to the version, i.e., with a updated user agent string and the newly-added web font.

2 MEASUREMENT PLATFORM

In this section, we introduce our measurement platform used to collect and generate two types of dataset: raw and dynamics. The raw dataset contains all the fingerprints including anonymized usernames, cookies, and IP addresses from the deployment websites; the dynamics dataset is processed by grouping fingerprints into browser instances and calculating the deltas.

2.1 Terminology Definition

In this part, we describe several terminologies that are used throughout the paper for those readers who are unfamiliar with them.

- Browser Instance and Browser ID. A browser instance is a piece of browser software installed on a certain operating system and a hardware device. For example, a Google Chrome Browser on a desktop is one browser instance and Microsoft Edge on the same device is another. We assign each unique browser instance an ID (called browser ID) and describe its makeup later in Section 2.3.1.

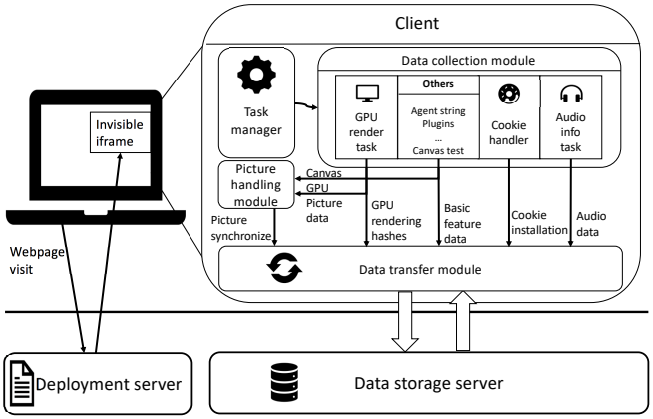


Figure 1: Architecture and deployment of our tool deployed at an European website for eight months.

- User ID. A user ID is an identifier for distinguishing one user from another, which is shared across different devices of the same user. In this paper, we adopt a hash value of the username as the user ID.
- Browser Fingerprint and Anonymous Set. A browser fingerprint (or for short fingerprint) is a set of features, such as user agent and font list, from a given browser instance. An anonymous set, a term widely used in prior works [25], is a set of browser instances with the same fingerprint. The smaller the anonymous set size is, the better quality the browser fingerprint is.
- Fingerprint Dynamics. A piece of fingerprint dynamics represents the change of one browser instance's fingerprint due to various reasons, such as browser updates and user actions.

2.2 Raw Dataset Collection

In this part, we describe our methodology in collecting a raw dataset with browser fingerprints, IP addresses and user IDs. We start from describing our tool and then present the tool's deployment.

2.2.1 Fingerprinting and Data Collection Tool. We implemented an open-source fingerprinting and data collection tool at a repository [5]. Our tool, as shown in Figure 1, has two main components: one data collection client and one data storage server. The client has a task manager that launches different tasks in parallel to collect a variety of features as documented by different prior works [8, 16, 25]. Then, the data transfer module of the client will encode the collected information and send it to the data storage server. Note that the data transfer module will check whether the information is already in the server's database—if yes, the module will just send a hash value to save the transmission overhead.

There are two properties of our tool that is worth mentioning. First, our tool is fast, which finishes collecting all the information within one second. Specifically, we divide data collection stages into many modules and run them in parallel. Then, the data collection module compresses the information transmitted between the client for efficiency consideration. Second, our tool utilizes only one third-party JavaScript library, i.e., three.js. The reason is that the website owner, citing their company policy, specifically asks us to avoid using third-party libraries, such as jQuery. Their concern is that the inclusion of third-party JavaScript library may introduce

unknown or under-controlled vulnerabilities. After many rounds of discussions, we mutually agree to keep the necessary one, i.e., three.js.

2.2.2 Tool Deployment. We deploy our tool at an European website from July 2017 to July 2018 to collect browser fingerprints. Our deployment can be divided into three stages and only the data collected from December 2017 and July 2018 in the Deployment Stage 3 is used in the study:

Deployment Stage 1: Deployment on Exit Webpage (two months). In the first stage, we install our tool on the least important webpage of our deployment website, i.e., the exit webpage that the user sees after clicking the log-out button. During this stage, we encounter and fix several bugs such as the use of old JavaScript features leading to console errors.

Deployment Stage 2: Deployment on 30% of Webpages (two months). In the second stage, we start to install our tool on 30% of webpages of our deployment website, including the login page and several other content pages. During this stage, we find that our server, deployed at Amazon, cannot handle the huge amount of traffic introduced from our deployment website, and therefore we have to increase both the memory and CPU capability of our server.

Deployment Stage 3: Deployment on All Webpages (eight months). In the last stage, we deploy our tool on all the webpages of our target website—the data collected during this stage is used in this study. The same as previous stage, we also increase our server capability to accommodate more traffic. Note that due to technical glitches, our data collection server was partially down during eight days in the first month. We also make two hot patches during our deployment: one on the 7th day to include the list of HTTP headers and the other on the 29th day to fix an error of "Accept" header collection in HTTP requests. That said, any fingerprint or statistics involving these two features only reflect data collected after these two days.

2.3 Dynamics Dataset Generation

In this part, we describe how to generate fingerprint dynamics from the raw dataset. The generation of the dynamics dataset has two steps. First, we represent each browser instance via browser ID and then group fingerprints based on browser instances. Second, we calculate the delta, using a diff operation, between each pair of consecutive fingerprints of the same browser instance: Those deltas are the dynamics dataset of our study.

2.3.1 Browser Instance Representation. We represent each browser instance with a special identifier, called browser ID. The generation of browser ID has two steps: (i) initial construction and (ii) processing of special cases. First, we construct an initial browser ID based on user ID and stable browser features, e.g., hardware-related ones including CPU class, device and OS, number of CPU cores, browser type and GPU information. Second, we link two browser IDs together if these two browser IDs belong to some exceptional cases observed by cookie instances. For example, if a mobile browser opens a webpage in the desktop mode, the observed browser type changes from mobile to desktop—this is one special case for such linking.

There are two things worth noting here. First, we adopt browser ID over cookies and user IDs alone due to the following reasons. We do not use cookies because a user may clear cookies and thus multiple cookie instances may map to one browser ID. Over 30% of browser instances in our dataset have cleared cookies at least once. Furthermore, we do not use user ID alone because a user may have multiple devices or use more than one browser to visit our deployment website. Over 15% of users in our dataset have used more than one browser for visits. Second, there are some software features, such as the support of localStorage and cookie, are also stable according to the cookie metrics but excluded from the browser ID. The reason is that the changes of these features are controlled by the users, thus being unpredictable. Furthermore, because we use cookies to gauge stability, the stability of such features may be influenced.

2.3.2 Diff Operation. In this part, we describe our diff operation that calculates the delta between two fingerprints of the same browser instance. Depending on the feature type, e.g., string, set and images, there are three different operations.

First, we will parse a string feature into ordered subfields and calculate the diff of each field. For example, the user agent is broken down into many ordered subfields, such as browser name, version, subversion, backslashes, parenthesis and even whitespaces. Note that we requires that subfields to be ordered because sometimes the sequence may also change, e.g., from "gzip, deflate, br" to "br, gzip, deflate". Furthermore, whitespaces may also be added or deleted, e.g., from "gzip,deflate" (no whitespace in Maxthon Browser 4.9.5.1000) to "gzip, deflate" (with whitespace in 5.1.3.2000).

Second, we represent a set feature just as a set and calculate the diff via two subtraction operations to obtain added and deleted elements. For example, the font list is obtained via querying each font and forming a set. We will calculate two subset: one for added fonts and the other for deleted fonts.

Third, we calculate the diff of two complex features, e.g., a canvas image, as a pair of two hashes. Note that it is possible to compute the pixel differences for such features. We did not adopt this approach because such delta does not contain much information, i.e., the change of the same pixel might not indicate the same update, and the computation involves heavyweight operation, slowing down the dataset generation.

2.3.3 False Negative and Positive Estimation. In this part of the section, we estimate the false negative and positive rates of our browser instance representation via browser ID. From a high level, our estimation is based on the appearance of cookies within or across different browser instances: Two browser instances with the same cookies are falsely separated, being a false positive; One browser instance with interleaved cookies should be separated into two instances, being a false negative. Then, we use the distributions of false positives and negatives among those browser instances that do not clear cookies to estimate those that do. Our overall estimation is that the false negative rate is around 0.3% and the false positive rate around 0.1%.

Now let us look at the details. First, we estimate false negative rate, i.e., two browser IDs should be linked together but not. Our investigation using cookies shows very few abnormal cases, i.e., 0.5% among all the browser instances in which two browser IDs

having the same cookie. Those cases, mostly due to a client providing fake user agent strings, are fixed via cookies, but there are 32% of browser instances that clear cookies (See Section 3.2.1). Therefore, we estimate that around 0.3% of browser instances among 32% browser instances may also have such abnormal cookie patterns.

It is worth noting that the cookie representation may sometimes also introduce false positives. For example, we observe that two iPads with different hardware features have the same cookie—this only happens once in our database. After some investigation, we believe that the user of these two iPads performs an iTunes backup so that our cookie is automatically transferred from the old iPad to the new one. In other words, these are indeed two browser instances.

Second, we estimate false positive rate, i.e., two browser IDs should be not linked together but actually are. Our methodology is based on the assumption that if two cookies appear together and are interleaved with each other in the time axis, this browser ID should be broken down into two. Note that this is different from a cookie deletion case, where deleted cookies will never show up again, or a private browsing, where one cookie persists but cookies in private browsing behave like deleted ones. There are 0.1% of browser instances with this pattern, thus categorized as false positives. We manually inspect these 0.1% of browser instances and think that it may be because users visit our deployment website using computers with exactly the same configurations, e.g., these in a computer lab.

3 DATASETS

In this section, we introduce both the raw and dynamics datasets. Note that per our agreement with the deployment website, we will share our dataset if other researchers reach out to us and sign a non-disclosure agreement (NDA), which confirms that (*i*) their use of our dataset is constrained in an academic setting, e.g., publishing academic papers, (*ii*) they will not release any potential private information contained in our dataset, and (*iii*) they will not give the dataset to any third-party.

3.1 Raw Dataset

We now introduce the raw data: it contains 7,246,618 fingerprints with 1,586,719 distinct values from 226 countries. Figure 2 shows the percentage of identifiable browser fingerprint when the size of anonymous set for each fingerprint increases. When the anonymous set size is 10, the identifiable browser percentage, including IP city, region and country as features, for our raw dataset is over 90%. Note that the identifiable browser percentage with the anonymous set size as one is relatively low because many browser instances visit our deployment website more than once, and we will show detailed breakdown regarding browser instances in later sections.

We also break down the identifiable fingerprint percentage based on different platforms and browsers in Figure 2. One interesting finding is that on desktop platform, Firefox is on par with other browsers in terms of fingerprintability, while Firefox on mobile platform is the most fingerprintable browser. The reason is that many mobile users will adopt the default browser, either Safari or Samsung Browser, in their cellphones. Therefore, the installation of another browser like Firefox is itself a fingerprintable feature. The

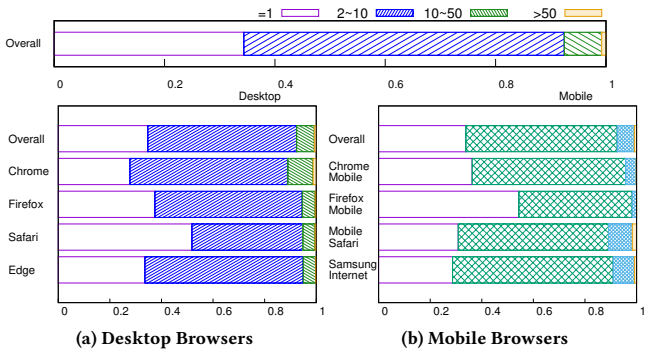


Figure 2: Percentage of identifiable browser fingerprints vs. the size of anonymous set in our raw dataset

same also applies to Chrome Mobile, which is less fingerprintable than Firefox Mobile but worse than Safari and Samsung Browser.

We further break down collected raw fingerprints by different features and show the distinct and unique values in Table 1. The distinct number means all the possible values for that feature, and the unique all the values that belong to only one fingerprint. Here is a brief description of those features below.

- HTTP Headers. HTTP headers contain many fingerprintable features, such as User-agent, Accept, Encoding and Language.
- Browser Features. Browser features include plugins, timezone, and support of different new browser functions, such as WebGL, localStorage, addBehavior (an IE feature), and openDatabase (a JavaScript-level database).
- OS Features. OS features include installed fonts like Arial, languages like Japanese and Chinese, and emojis (i.e., part of Canvas Images in Table 1) like a smiling face. We rely on two side channels documented by prior works [3] to detect the list of fonts and installed languages.
- Hardware Features. Hardware features include information about GPU, audio cards, screen and CPU. Modern browsers provide some APIs to access hardware information, such as CPU class (e.g., x86), GPU vendor (e.g., NVIDIA), and audio card information (e.g., number of channels).
- IP Features. IP addresses are not included in browser fingerprinting because a user may move a device from places to places. For the reason of completeness, we abstract some information from IP addresses, such as IP city, region and country.
- Consistency Features. Consistency features [8] refer to whether our script can obtain consistent information on a certain feature via different methods. For example, we can obtain OS and browser information from both user agent and JavaScript navigator, and then check the consistency between these two.

Note that the list of fonts collected by JavaScript via a side-channel is the most fingerprintable among all the features in terms of distinct and unique values. After that, both user agent and the list of plugins, especially the latter, also contribute a lot to the overall fingerprint. The user agent contains many information, such as platform and browser type, which makes itself a big fingerprintable vector. As for the plugin list, if a user installs a plugin, it is more or less unique as compared to those who do not have plugins. It is

Table 1: Statistics of different features used in the dynamics dataset ("Distinct #" the number of distinct values for fingerprint or dynamics and "Unique #" the number of values that only appear once. A feature with an indent means that the feature is a subset of the top-level one.)

Feature Names	Static Values		Dynamics	
	Distinct #	Unique #	Distinct #	Unique #
HTTP Headers	195,845	136,256	18,180	12,725
User-agent	41,060	23,116	9,628	6,152
Browser	64	8	53	30
OS	20	3	23	5
Device	3,378	1,210	277	226
Accept	9	1	4	0
Encoding	26	3	26	8
Language	14,214	9,191	1,939	1,458
Timezone	38	0	314	112
HTTP Header List	344	126	126	66
Browser Features	17,036	14,362	1,037	795
Plugins	16,633	14,032	984	773
Cookie Support	2	0	2	0
WebGL Support	2	0	2	0
localStorage Support	2	0	2	0
addBehavior Support	1	0	0	0
openDatabase Support	1	0	0	0
OS Features	193,843	150,280	16,605	12,793
Language List	1,181	597	452	303
Font List	115,128	88,448	6,763	5,524
Canvas Images	14,006	8,654	7,989	5,524
Hardware Features	75,462	44,708	4,871	3,210
GPU Vendor	26	1	2	1
GPU Renderer	5,747	1,743	705	552
GPU type	4,943	1,436	214	130
CPU Cores	29	3	28	12
Audio Card Info	114	23	225	62
Screen Resolution	139	32	273	149
Color Depth	6	0	10	2
CPU Class	5	0	4	3
Pixel Ratio	1,930	1,207	3,030	1,936
IP Features	28,636	8,720	122,612	84,232
IP City	27,261	8,112	121,565	83,445
IP Region	2,446	239	16,376	9,947
IP Country	226	9	1,627	779
Consistency Features	13	1	19	4
Language	2	0	2	0
Resolution	2	0	2	0
OS	2	0	2	0
Browser	2	0	2	0
GPU Images	4,152	2,719	2,810	1,499
Overall (excluding IP)	960,135	852,987	89,397	66,857
Overall	1,586,719	1,447,004	359,374	306,554

worth noting that IP information, such as city, region, and country, also provides a considerable amount of information.

3.2 Dynamics Dataset

Our dynamics dataset contains 1,329,927 distinct browser instances: 661,827 of them visit the deployment website for more than one time, which produces 960,853 pieces of dynamics information. In the rest of the section, we first present statistics of browser instances and then statistics of dynamics.

3.2.1 Statistics of Browser Instance. We now show some statistics of browser instances in the dynamics dataset.

- *User ID vs. Browser ID vs. Cookie.* The top bar of Figure 3 shows that approximately 86% user IDs (anonymized usernames) map to only one browser ID while the rest maps to more than one

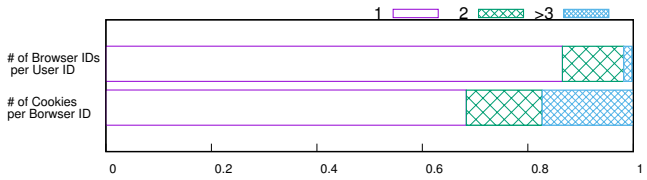


Figure 3: A Breakdown of the Number of Browser IDs per User ID and the Number of Cookies per Browser ID (For example, the purple bar with no fills in "# Browser IDs per User ID" means the percentage of all user IDs that have one browser ID.).

because those users visit our deployment website from more than one browser instance. The bottom bar of Figure 3 shows that 68% of browser instances have only one cookie; to the opposite, about 32% of browser instances have more than one cookie. As stated, our manual investigation with controlled testing of Safari Browser shows that intelligent tracking prevention and private browsing are the major reasons of clearing cookies.

- *Browser instance visits over time.* Figure 4 shows the number of browser instances broken down by first-time and returning visitors across our measurement period. A browser instance is marked as a returning visitor, if its browser ID has been seen in our dataset before. The first thing worth noting is that the number of total visits by browser instances in the first three months is higher than the rest. The reason is that our deployment website in general has more visitors during the holiday season, which leads to the visit number decline in our dataset during the remaining months. Second, returning browser instances make up almost half of all the visitors each day—this fact indicates that our deployment website has a considerable amount of loyal users for us to collect enough dynamics data.
- *Browser instances broken down by browser and OS types.* We also show the number of browser instances broken down by browser types in Figure 5 and by OS types in Figure 6. Figure 5 shows that our visitors are well distributed into different browser types on both mobile and desktop platforms, i.e., being a good representation of the Internet users; Figure 5 shows that Microsoft Windows is still the mostly used OS in our dataset and the next comes with iOS, which is used in both iPhones and Apple computers. Figure 6 shows that the percentage of browser instances using Android OS is on par with iOS. The number of Ubuntu and Windows Phones is too small to be shown in the graph.
- *Fingerprint stability per browser instance.* We break down browser instances (browser IDs) based on the number of visits and the number of dynamics in Figure 7. When a browser instance visits our deployment website for three or four times, about half of browser instances remain stable without fingerprints changed. The percentage keeps decreasing as the number of visits increases and then stays at about one third.

3.2.2 Classification of Fingerprint Dynamics. We classify fingerprint changes into three categories based on their causes and show them in Table 2:

- Browser or OS Updates. Browser or OS updates, taking up to about 30% of total changes, refer to the client browser or OS updates to a new version. Such an update may lead to a change

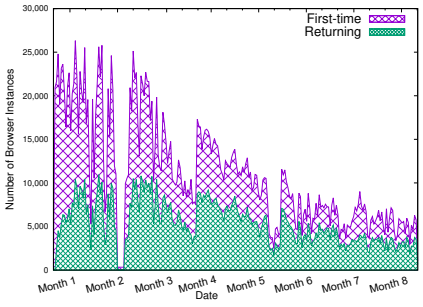


Figure 4: The number of first-time and returning browser instances over the entire deployment period

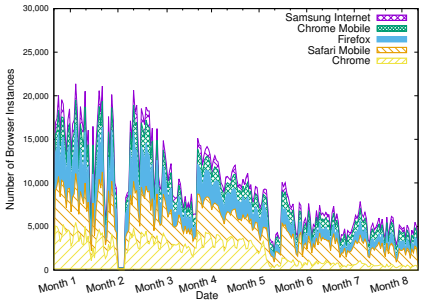


Figure 5: The number of browser instances broken down into different browser types

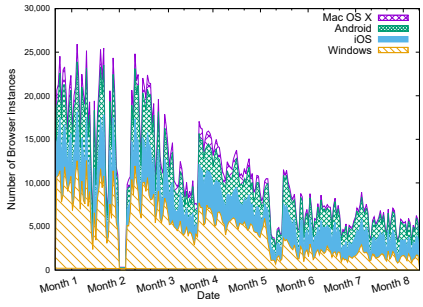


Figure 6: The number of browser instances broken down into different OS types

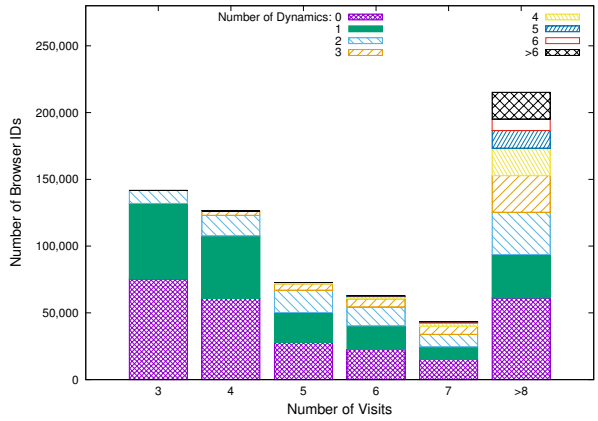


Figure 7: A breakdown of the number of browser IDs based on the number of dynamics and the number of visits (For example, the solid, green bar above 3 on the x-axis indicates the number of browser IDs satisfying the following two conditions: (i) a browser instance visits our deployment website for only three times and (ii) the fingerprint of that browser instance changes only once—i.e., containing only one piece of dynamics information.)

in the user agent string and other correlated features, such as canvas rendering and the font list. We first look at OS updates: iOS updates is the single largest portion, i.e., over 95%, of all the updates, because all the subversions of iOS are included in the user agent string. As a comparison, browser updates spread more evenly across different browsers based on their use percentage. It is worth noting that the percentage of browser instances with OS and browser updates is relatively small, i.e., only 8.1% and 13.81% respectively. That is, many browsers or OSes are not constantly updated, which may lead to corresponding security issues.

- User Actions. Some user actions may lead to fingerprint dynamics, e.g., zooming in/out of the current page changes the screen resolution provided by the browser. It is interesting that only 13.4% of total browser instances have user-action-related dynamics as opposed to 31.07% of total dynamics: Such a big gap shows that a large portion of users do not perform actions that can change fingerprint, but if a user does perform an action, it is very likely that she may do it again in the future. One big portion of dynamics related to user actions is timezone change, taking up 40.49% of total browser instances in this category, which is caused by a user movement from one location to another. The reason that timezone change happens often is that our deployment website locates in Europe and many users travel from one country to another for work.

- Environment Updates. When other software co-located with the browser instance is updated, browser fingerprints may change as well. First, some software updates, such as Microsoft Office and Adobe Acrobat Readers, may introduce new fonts to the OS—about 6.74% of environment updates belong to such category. Second, one big subcategory of environment updates is due to the change of emojis—87.6% of canvas rendering result updates are caused by rendering emojis rather than texts. Lastly, other environment factors, such as audio card information, system languages, and color depth may change as well. It is also worth noting that the percentage of browser instances with environment updates, i.e., 5.57%, is also the smallest compared with other causes, although the percentage of dynamics with environment updates is similar to the one of OS updates. The reason is that environment updates have to happen when certain environment, e.g., a specific type of software, co-exist with the browser. Take Adobe Software for example—if someone does not use Adobe Acrobat Reader, such environment updates will not exist for that browser instance.

We also listed composite changes that lead to fingerprint changes. The percentage of such composite changes aligns with the percentage of each single category because all the changes are independent. For example, user actions and browser updates are two major categories and therefore the combination of these two is also the largest category among all the possible combinations. It is worth noting that the combination of browser and OS updates are not much, because many browser and OS updates, especially on iOS platform for Safari, is related and counted as OS updates already.

3.2.3 Breakdown of Dynamics by Features. We break down the dynamics by different features and also show them in Table 1 under the dynamics column. In total, we have observed 359,374 pieces of distinct dynamics information; interesting, 306,554 of them, i.e., 85%, are unique. Additionally, there are several things worth noting, especially when comparing with the static values of each feature.

Table 2: A Breakdown of Fingerprint Changes (The total percentage of fingerprint changes adds up to 100%, and the union of all browser instances equals to the percentage of browser instances with fingerprint changes).

Operation Category	% of Changes	% of Browser ID
OS Updates	+11.26%	8.10%
iOS	+11.26%×96.31%	×95.67%
Android	1.71%	2.20%
Mac OS X	1.37%	1.60%
Windows	0.54%	0.50%
Others	0.07%	0.03%
Browser Updates	+19.69%	13.81%
Chrome	×39.01%	×34.67%
Firefox	16.95%	19.39%
Chrome Mobile	26.28%	26.25%
Samsung Internet	8.09%	9.40%
Opera	2.67%	2.45%
Edge	1.94%	2.53%
Firefox Mobile	1.76%	1.96%
Safari	1.20%	1.37%
Others	2.1%	1.98%
User Actions	+31.07%	13.40%
Change timezone	×19.43%	×40.49%
Private browsing mode	41.01%	33.85%
Zoom in/out webpage	17.27%	11.37%
Enable/disable Flash	13.63%	7.02%
Fake supported languages	6.00%	8.10%
Fake screen resolution	2.62%	3.76%
Switch monitor/change resolution	2.45%	2.80%
Browser/OS inconsistency	1.14%	1.3%
Request desktop website	38.52%	47.18%
Others (e.g., fake agent string)	61.48%	52.82%
Install plugins	1.27%	1.12%
Enable/disable LocalStorage	0.64%	1.19%
Enable/disable Cookie	0.41%	0.71%
Environment Updates	+11.91%	5.57%
Software Updates (fontlist)	×6.74%	×8.06%
MS Office	27.08%	36.91%
Adobe Software	33.39%	23.79%
Office and Adobe Software	1.04%	1.35%
Others	38.49%	37.95%
Update Canvas rendering	53.38%	53.20%
Emoji update	87.60%	87.15%
Text update	12.40%	12.85%
Audio update	39.83%	40.57%
HTTP Header Language update	1.77%	2.68%
System Language update	0.74%	0.69%
Screen color depth update	0.22%	0.40%
GPU Render update	0.20%	0.32%
Browser Updates + User Actions	+10.19%	8.78%
OS Updates + User Actions	+5.17%	4.64%
Browser + Environment Updates	+1.83%	1.54%
Other Combinations	+8.88%	6.48%
Total	=100%	62.32%

First, the list of fonts, a highly fingerprintable feature with many distinct and unique values, stays relatively stable in terms of dynamics. We only observe 6,763 distinct dynamic values as opposed to 115,128 distinct static ones. That is, the list of fonts is a relatively good feature for browser fingerprinting. Interesting, even if the list of fonts changes, it is highly likely that the changes are unique as well: 5,056 out of 6,763, i.e., 74.8% of dynamics, is unique, which means that font update will also reveal the client browser with high probability.

Second, these features that are influenced by user actions have more dynamic values when compared with their static ones. Such features include IP features, timezone, screen resolution, and pixel ratio. Take screen resolution—which is influenced by a user zooming in or out the webpage—for example. It has 139 static values but 273 dynamics. Similarly, timezone has 38 static values but 314 dynamics. The reason is that when a feature is influenced by users, the change is usually bi-directional and has less restriction. That is, the value of that feature may change from one value to any in the set. For instance, users are free to move from one location to any place in the world, thus causing a possible dynamic value for timezone and IP-related locations. As a comparison, the dynamics for screen resolution has more restrictions. Although users are free to zoom in or out a web page, the screen ratio stays the same after such operation. Therefore, the dynamic to static value ratio for screen resolution is also smaller than that of timezone.

(a) Canvas Rendering Result on Samsung Browser **(b) Pixel difference between version 5.4 and 6.2 (highlighted in red)**

Figure 8: Samsung Browser version 6.2 introduces a new emoji that is also visible from a Google Chrome Browser co-installed with the Samsung Browser (The difference between those two emojis is the red-color part, i.e., a smiling face emoji shown in Subfigure (b))

Third, hardware-related features, such as these used in browser ID, are relatively stable, i.e., with very few dynamics. All the dynamics are special cases in which we need to link two browser IDs together as we mentioned in the browser ID generation.

Lastly, the number of dynamics is usually a fraction of, or on par with, the static values for the rest of features. The reasons are twofold. (i) Most features are stable, i.e., many static values are not involved in a dynamic one. (ii) Some fingerprint changes are restricted, e.g., an unidirectional one. Take an OS update for example, which happens only from a lower version to a higher version. (We do not observe that anyone downgrades their OS in our dataset.) That is, two static values map to only one dynamic one.

4 INSIGHTS

In this section, we present several insights when observing our raw and dynamics dataset, and then give some advices based on each insight to browser vendors, users or fingerprinting tool developers.

Insight 1: Browser fingerprints, particularly the dynamics, reveal privacy- or security-related information.

Insight 1.1: System-provided emojis may leak security patches involved in OS updates.

We find that system-provided emojis may be used to infer security related patch, such as those in OS updates. In particular, we list two cases in which browser or OS updates lead to emoji changes—i.e., in other words, such emoji changes can be used to infer corresponding software updates involving security patches.

- Emoji changes in Mobile Google Chrome caused by Samsung Browser Update. A Samsung Browser update is observable in Google Chrome canvas rendering results. Specifically, Samsung Browser 6.2 introduces a new emoji that has a slight change of the smiling face as shown in the pixel-by-pixel difference of

Figure 8. Such update is also observable if Google Chrome renders the smiling face emoji on a canvas. That is, to summarize it, if one observes a canvas rendering update like Figure 8 in Google Chrome Mobile, we can infer that the user updates his Samsung Browser installed on the same device to 6.2, and otherwise not. We find 2,298 Chrome instances in our dataset, which leaks such private information.

- Emoji changes in Desktop Google Chrome caused by Windows 7 Update. One Microsoft Windows 7 update on April 22, 2014 installs a set of new emojis to the OS by introducing IE 11, and such emoji updates are observable from another browser, such as Chrome. We only observe 9 browser instances with such emoji updates, because the update was released back in 2014. Interestingly, we also observe 6,968 browser instances with the old emoji, i.e., they have not applied that specific old update, leading many potential security vulnerabilities, i.e., those that are found after that update, unpatched. Note that browsers on Windows platform only indicates the big version, i.e., 7, 8, or 10, of OS. That is, such update information is supposed to hide from a website visited by the user.

Advice 1 [Browser Security]: Browsers should provide their own emojis to avoid leaking whether security patches are applied.

Insight 1.2: System-provided fonts may leak updates and installations of software, such as Microsoft Office.

System-provided fonts can be used to infer software updates and installations. If knowing software updates and installation, an attacker can launch targeted attacks, such as macro malware aiming at Microsoft Word. We now list several examples of such font-related inference below:

- Font changes caused by Microsoft office update. Our reasoning results show that one particular added font in any browser can reveal the information about a Microsoft Office update. Specifically, the release of three versions of Microsoft Office, i.e., Version 1711 (Build 8730.2175), Version 1708 (Build 8431.2153), and Version 1705 (Build 8201.2217) on January 9, 2018, will add a new font called "MT Extra", which is observable in a browser fingerprint. Therefore, the addition of an "MT Extra" in early 2018 is a strong indication that the device has installed Microsoft Office and updated it accordingly. We find that 1,199 browser instances added the font "MT Extra". Note that this is just a subset of browser instances that applied the update because if the OS has already installed "MT Extra", e.g., by other software before the update, we will not observe the change.
- Font changes caused by Microsoft Office. Apart from the previously mentioned Office update, the installation of Office itself also introduces new fonts. We find 7 browser instances that are related to the installation of Microsoft Office Pro Plus 2013, i.e., reflected in a font list change. Additionally, we observe 50,869 browser instances installed with Microsoft Pro Plus 2013, because their font list contains corresponding fonts installed by Microsoft Office.
- Font changes caused by WPS Office and LibreOffice. Both WPS Office, an office suite developed by Kingsoft, and LibreOffice, a free and open-source office suite, add a new list of fonts to the system that lead to a fingerprint change. Note that WPS office also slightly changes the color of the emoji rendering.

Advice 2 [Browser Security]: Browsers should ship their own fonts, such as Web fonts, like what Tor Browser does to avoid leaks of software updates and installations.

Insight 1.3: The rendering effects of GPU images can be used to infer masked hardware information.

The rendering behaviors of GPU can revealmasked GPU information. Specifically, based on GPU images collected from other browsers, our correlation analysis finds that 32% of distinct Firefox GPU images can be uniquely mapped to one renderer and vendor, and 38% can be mapped to less than three renderers and vendors. It is interesting that the inference accuracy for certain GPU types, especially these dedicated GPU vendors, are very high, because these GPU rendering behaviors are very different from others when they try to pursue a high rendering quality. For example, the inference accuracy for NVIDIA GeForce series is usually larger than 90%, with GTX 970 as 95.5%. Mali and PowerVR GPUs are very unique as well, with 96.2% and 92.4% inference accuracy respectively. On the contrary, the inference accuracy for low-end, integrated GPUs, such as AMD and Intel ones, are relatively low, which are 20.8% and 57.4% respectively.

Advice 3 [Browser Privacy]: Browsers, such as Firefox, should change canvas rendering results as what Wu et al. [45] do when masking GPU information.

Insight 1.4: IP address change can be used to infer network status, e.g., the use of VPNs or proxies.

Specifically, we can calculate the velocity of the browser instance based on the IP information, such as the latitude and longitude provided by the public database, between two consecutive visits. If the velocity is larger than a threshold, say 2,000 km/h, which is impossible even by plane, we can consider that the browser instance adopts network services, such as proxy and VPN, to visit our deployment website between these two visits. Our evaluation shows that the velocities of most browser instances are small, i.e., less than 150 km/h. There are no browser instances in our database moving between 150 km/h and 2,000 km/h—this is probably because usually the proxy or VPN is located far from the user. We have observed 2,916 browser instances moving over 2,000 km/h, which are considered as using VPN or proxy service. We look at manually some cases and verify that they are indeed using network service. For example, one user was using a Russian IP address at Kaluga; one day later, her IP address was changed to one at Lagos, Nigeria, Africa; and then two hours later, her IP address went back to the first one. The moving speed is way beyond 2,000 km and her second IP address, after manual verification, belongs to a public VPN service.

Advice 4 [User Privacy]: Users may want to avoid visiting a website with and without VPN/proxy service at the same time.

Insight 2: The F1-score and matching speed of prior evolution-aware fingerprint work degrade significantly in a large-scale setting.

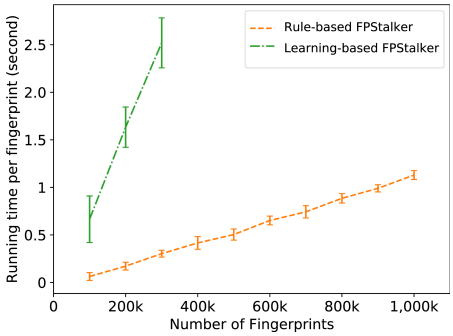


Figure 9: Matching Time of FP-Stalker against One Fingerprint (Note that matching time greater than 100 ms is considered unacceptable because ads real-time bidding (RTB) requires that an advertiser provides a decision under 100 ms [42, 48], a hard limit enforced by many ad exchange networks like Google)

We evaluate state-of-the-art evolution-aware fingerprinting tool, FP-Stalker, using the dataset collected in our measurement study. All the experiments are performed on a powerful server with 192 GB RDIMM 2666MT/s Dual Rank memory and Intel® Xeon® E5-2690 v4 2.6GHz CPU. There are two variations of FP-Stalker, rule-based and learning-based. We adopt all the original rules from the paper and retrained the learning-based FP-Stalker as the F1-Score of the original model is very low (smaller than 50%) on our dataset.

We look at two important metrics of FP-Stalker:

(i) Matching Speed. Figure 9 shows the average matching time of FP-Stalker against one fingerprint, which increases linearly as the number of fingerprints. We would like to point out that the matching speed of FP-Stalker, no matter rule- or learning-based, is unacceptable in this large-scale setting. The reason is that many ad exchange networks like Google requires that an advertiser provides a decision under a hard limit, which is 100 ms [48].

(ii) F1-Score, Precision and Recall. Figure 10 shows the precision, recall, and F1-Score of FP-Stalker as the number of fingerprints increases: All three numbers drop linearly. We now list some false positive (FP) and negative (FN) examples below:

- FN: A desktop page on a mobile device. FP-Stalker fails to link those two fingerprints in Figure 11 (a), as the user agent changes drastically from a mobile Chrome (Fingerprint 1) to a Linux Desktop (Fingerprint 2).
- FN: Storage place disabled on Chrome. Figure 11 (b) shows an FN example of storage places, such as cookies and localStorage, are disabled on Chrome, which lead to a change from Fingerprint 1 to 2.
- FP: Two browser instances with different CPU cores. Figure 11 (c) shows an FP example of two browser instances with almost exactly the same fingerprint but different CPU cores. This change is very unlikely to our human being, but will be considered as possible by FP-Stalker.
- FP: Two browser instances with different device types. Figure 11 (d) shows an FP example of two browser instances with just different device types. Again, this change is small, i.e., from J330 to G920, but very unlikely.

Table 3: Case Studies on Feature Correlation with Browser or OS Updates (Emoji type means a redesign of emoji, and emoji rendering is some subtle rendering detail changes; text width means the width of text rendered in browser canvas, and text detail is some subtle text rendering detail changes.)

Update	Platform	Correlated Feature: Changed Value
Browser Update on Mobile Phone		
Mobile Safari 10→11	iOS	Canvas (C): Emoji rendering
Mobile Safari 11→12	iOS	Font (F): Remove two fonts
Samsung 5→6	Android	C: Emoji rendering
Samsung 6→7	Android	C: Text width and emoji rendering
Mobile Firefox 56→57	Android	C: Text width
Browser Update on Desktop		
Safari 10→11	Mac OS X	F: Remove/add fonts
Safari 10→11	Mac OS X	C: Emoji rendering
Firefox 60→61	Ubuntu	C: Emoji type
Chromium 62→63	Ubuntu	Plugin (P): Remove one plugin
OS Update on Mobile Phone (* means any version lower than the update target)		
Android *→4.4.2	Android	C: Samsung emoji rendering
Android *→8.0.0	Firefox	C: Text width and emoji type
Android *→8.0.0	Samsung	C: Text width and emoji rendering
Android *→8.0.0	Chrome	C: Text detail
iOS *→10.3.3	Safari	C: Emoji rendering
Blackberry OS *→10.3.3	Webkit	C: Text detail
OS Update on Desktop		
Windows *→10	Maxthon	C: Text width and emoji type
Mac OS X *→10.10.4	Safari	C: Emoji rendering
Mac OS X *→10.13	Firefox	C: Text width

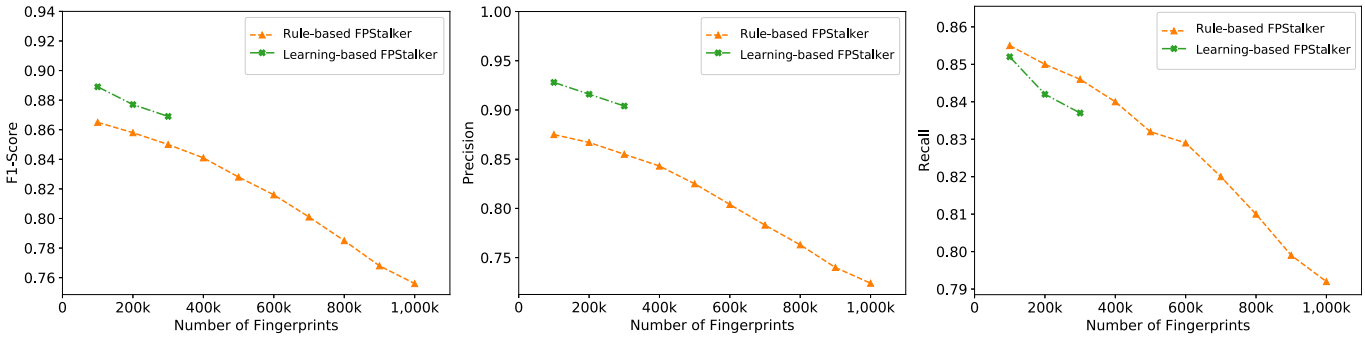
Advice 5 [Better Fingerprinting Tool]: Existing fingerprinting tools need to consider semantics of browser dynamics to improve its precision, recall and F1-Score.

Advice 6 [Better Fingerprinting Tool]: Existing fingerprinting tools may consider caching to improve its matching speed and meet the real-time requirement.

Insight 3: The dynamics of some browser features are correlated although the features themselves are not.

We show that although some features are not correlated directly, the dynamics of those features may be implicitly. Our methodology of finding such correlation is as follows. We first rank all the dynamics based on their popularity, i.e., the total number of appearance, and then find dynamics in which two features come together. We consider these two feature dynamics are potentially correlated if these two features either do not come separately in the dynamics database or appear less popular than the combined one. We then manually inspect these two feature dynamics to understand whether they are correlated. Here are some examples of such implicit correlations:

- Example 1: Cookie disabling/enabling is correlated with localStorage in Chrome Browser. That is, when cookie is enabled or disabled in Chrome, localStorage will change as well. In total, we have observed 347 Chrome instances that disable cookie and localStorage together and 226 that enable them together. The reason is that Chrome provides a single checkbox to disable or enable both cookie and localStorage; interestingly, the disabling/enabling of cookie and localStorage is not correlated in



(a) F1-Score (b) Precision (c) Recall

Figure 10: F1-Score, Precision and Recall of FP-Stalker for Top 10 Prediction (Note that we run both learning- and rule-based FP-Stalker for 240 hours, which is ten full days; learning-based FP-Stalker is not scalable to a large dataset as acknowledged in the paper as well).

Fingerprint 1:
```
User Agent: Mozilla/5.0 (Linux; Android 9; SM-N960U)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/77.0.3865.92 Mobile Safari/537.36
...
```

Fingerprint 2:
```
User Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/77.0.3865.92 Mobile Safari/537.36
...
```

(a) A desktop page on a mobile browser

Fingerprint 1:
```
...
Support of Cookies: Yes
Support of localStorage: Yes
...
```

Fingerprint 2:
```
...
Support of Cookies: No
Support of localStorage: No
...
```

(b) Storages Disabled on Chrome

Fingerprint 1:
```
...
CPU Cores: 4
...
```

Fingerprint 2:
```
...
CPU Cores: 2
...
```

(c) Different CPU Cores

Fingerprint 1:
```
User Agent: Mozilla/5.0
(Linux; Android 7.0; SAMSUNG SM-J330F Build/NRD90M)
AppleWebKit/537.36 (KHTML, like Gecko)
SamsungBrowser/6.2 Chrome/56.0.2924.87
Mobile Safari/537.36
```

Fingerprint 2:
```
User Agent: Mozilla/5.0
(Linux; Android 7.0; SAMSUNG SM-G920F Build/NRD90M)
AppleWebKit/537.36 (KHTML, like Gecko)
SamsungBrowser/6.2 Chrome/56.0.2924.87
Mobile Safari/537.36
...
```

(d) Two Browser Instances on Different Devices

Figure 11: False Positives and Negatives of both Rule- and Learning-based FP-Stalker (*(a)* and *(b)* are false negatives, as they belong to the same browser instance but are not linked; *(c)* and *(d)* are false positives, as they are from different browser instances but are linked together. We skip the same features between each 1 and 2 pair).

Firefox Browser, because there are two places to perform these two actions.

- Example 2: The change of DirectX API levels in Firefox is correlated with Firefox updates among 57–60 on certain devices. Specifically, we find the DirectX API level is downgraded to 9EX when Firefox is updated to 58 or 59 on certain devices, and then the level is back to 11 when Firefox is updated to 60. We suspect that Firefox 57, a relatively buggy version [9], has some problem using DirectX 11 on certain devices and therefore it falls back to DirectX 9EX on Firefox 58 and 59. Then, Firefox 60 fixes some bugs and therefore reuses DirectX 11.
- Example 3: The change of DirectX API levels in Chrome is correlated with Audio Card Sample Rate. Specifically, we find that when Chrome's GPU renderer is updated from Direct3D 9EX to 11 on certain devices, the sample rate of its audio card will also be updated from 44,100 to 48,000. The reason is that probably Chrome adopts DirectX to manage audio card and therefore when DirectX is updated, audio card information is as well.

Except for those implicit correlations, Table 3 also shows correlations related to browser or OS updates. There are three major types of correlated features: canvas rendering results, font list and plugin list. The canvas rendering results is the most common correlation, because many browser and OS updates include new emojis or text rendering. Specifically, we classify the dynamics in canvas rendering results into four subtypes: text width, text details, emoji types and emoji rendering. Text width means the width of the text part of canvas rendering, which may changes if one letter is rendered thinner or thicker; text details means some texture details of the letter rendering; emoji types means the introduction of a new emoji type; emoji rendering means some small changes, such as smoothing of emojis.

Advice 7 [Better Fingerprinting Tool]: Existing fingerprinting tools may include implicit or explicit feature correlations to improve linking performance.

Insight 4: The timing of some fingerprint dynamics are correlated with real-world events, such as the release of browser or OS updates.

Specifically, we show such trends in Figure 12, where the x-axis is our deployment period and y-axis is the percentage of browser instances with corresponding browser update related dynamics. For example, 66 in Chrome sub-figure refers to fingerprint dynamics that are related to an update to Chrome version 66. We marked each

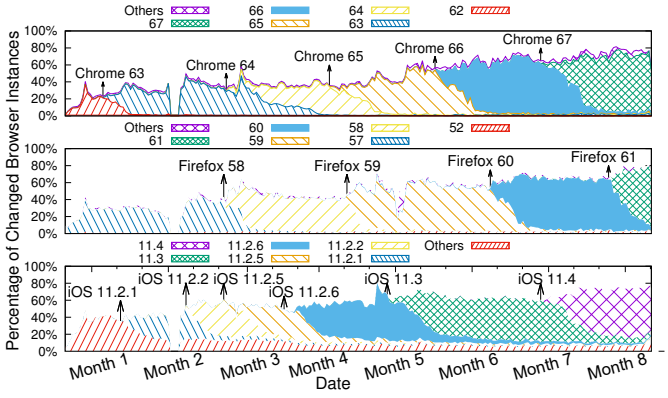


Figure 12: Percentage of browser instances with dynamics related to browser updates over the entire period of our deployment

important real-world event during our deployment period, such as Chrome updates between 63 and 67, Firefox updates between 58 and 61, and Safari updates between 10.1.2 and 11.1.

There are two things worth noting. First, after each browser release, there emerges a peak of fingerprint dynamics that lead to an update to the latest version. This trend is the same for all three browsers marked in the figure. Such updates are not immediate, which usually take months to finish. Second, the updates for Safari is usually slower than the ones for Firefox and Chrome. The reason is that Firefox and Chrome updates are automatic—a user just needs to restart the browser and update their browsers. As a comparison, Safari updates require a user to click several buttons in app store and therefore some users are reluctant of doing so.

Advice 8 [Better Fingerprinting Tool]: Existing fingerprinting tools may consider the timings of real-world events to improve linking performance.

5 DISCUSSIONS

We discuss several commonly-raised issues, such as ethics and feature inconsistency, in this paper. First, we obtained approval from our Institutional Review Board (IRB) prior to conducting the research. Our collected data via the deployment website may contain human information. Specifically, the deployment website has an agreement, i.e., a div element pointing to a legal document, stating that the website may collect user information including browser fingerprints and asking the users for their consent. During the sign-up stage, the users will also see an additional webpage asking for their consent of collecting fingerprint information. Since all the fingerprinting data are accompanied with an anonymized user ID, all the users in our study have at least seen the agreement twice and agreed to be collected. That said, all the ethic issues are handled through the deployment website via a standard procedure and the collection process obeys the EU privacy legislation, e.g., GDPR, which states that websites need to get visitors' consent to store or retrieve information on a computer, smartphone or tablet.

Second, feature inconsistencies have minimum impacts on our measurement study due to their small numbers. Specifically, a browser may provide a piece of false, or called inconsistent, information in our study and our tool actually adopts this type of inconsistency as a feature in the fingerprint. The number of such inconsistencies is very small (less than 1% of browser instances). Now let us look at some of these reasons that lead to inconsistencies. First, the user may want to request a different version of a web page on the device, e.g., a mobile device requesting for a desktop page. This is the major reason in our measurement study that leads to inconsistency and our analysis has already considered such scenarios. Second, a browser, such as Tor Browser and Brave Browser, or a privacy-preserving browser extension may conceal the browser's identity due to privacy reasons. The number of users having these browsers or extensions are relatively small, thus having minimum impacts on our measurement results. We will leave a measurement of such privacy-preserving tools as our future work.

Third, we discuss the limitations of adopting browser IDs as the ground truth. Although we consider it as a big improvement over user ID and cookies, the adoption of browser ID leads to false positives and negatives. For example, if one user has two identical devices, we will falsely assign the same browser ID to browsers on those two devices. For another example, there might exist some rarely happened user agent changes that are not captured during our study, leading to false negatives.

Fourth, we discuss attack traffic on the deployment website—This is an orthogonal problem to the paper. Because the recorded traffic belongs to users that are logged into the websites, the possibility of attack traffic, such as credential stuffing, is low. Specifically, most traffic related to credential stuffing is trying to log in with account credentials instead of visiting the website normally.

Lastly, we talk about the usage of browser fingerprints in the real-world. Although it is well known that browser fingerprints can be used for web tracking, which may violate user privacy, recent adoption of browser fingerprinting is sometimes to the opposite in the realm of two factor authentication and bot detection [4]. The intuition is simple: Browser fingerprints, just like cookies, have two sides: one for tracking and the other for authentication. In this measurement paper, we took a neutral view on how browser fingerprints are used, but focused on how fingerprints may change over time. It is the responsibility of those who deploy fingerprinting to decide its usage and we give advices on both sides in the paper.

6 RELATED WORK

In this section, we discuss related work including the closest one discussing fingerprint evolution/dynamics, web tracking and anti-tracking.

6.1 Fingerprint Evolution/Dynamics

FP-Stalker [40] is the first work that considers fingerprint evolution and designs an approach to link different browser fingerprints even if they evolve over time. The major contribution of FP-Stalker is the design and implementation of novel fingerprint linking algorithms, i.e., both rule-based and learning-based approaches. To validate their algorithms, FP-Stalker involves a relatively small dataset with 1,905 browser instances—as opposed to over 1 million in our paper and over 300,000 if we only count users visiting the deployment

website for more than seven times, i.e., following their criteria—collected from a group of users who install their browser extension.

After that, Pugliese et al. [37] also conduct another small-scale study with 88,088 fingerprints belonging to 1,304 users to understand users' trackability. Together with the study, Pugliese et al. propose a method, called feature stemming, to improve feature stability—which performs better than FP-Stalker on the FP-Stalker dataset. There are two potential issues of feature stemming. First, we believe that although feature stemming improves stability, there are still dynamics that need additional linking. Consider the example of a user requesting a desktop page on a mobile device. The user agents of two visits are drastically different, which cannot be captured by feature stemming. Second, feature stemming, e.g., stripping off version substrings, increases the anonymous set size of fingerprints, thus reducing fingerprintability in general.

As a general comparison with prior works on fingerprint evolution, our measurement study is in a much larger scale and also makes observations related to privacy and security, e.g., the leaks of software updates. Furthermore, our measurement study shows that both the learning- and rule-based FP-Stalker performs poorly in terms of F1-Score and matching speed in our large-scale dataset.

6.2 Web Tracking

We present related work in web tracking from two perspectives: cookie or super cookie-based and then browser fingerprinting. As a general comparison, our measurement study is the first work that classifies and measures dynamics in browser fingerprinting, a special, second-generation web tracking, and then draws interesting observations, such as dynamics-related privacy leaks.

6.2.1 Cookie or Super Cookie based Tracking. Many measurement studies have been proposed before on the effectiveness or severeness of Web tracking in general, such as these based on cookies or other server-set identifiers. For example, Roesner et al. [38] performs a comprehensive measurement study on web tracking and proposes a classification framework. Lerner et al. [26] conduct an archaeological study by measuring web tracking from 1996 to 2016 in Internet time machine. Soltani et al. and Ayenson et al. measure how tracking companies can use non-cookie based stateful tracking to regenerate deleted cookies [13, 39]. Metwalley et al. [28] adopt an unsupervised method to detect user identifiers that could be adopted for tracking purpose. Krishnamurthy et al. [21–24] gauge the harm of web tracking and conclude that trackers may obtain personal information, such as username and emails.

6.2.2 Browser Fingerprinting. Browser fingerprinting is the second generation of web tracking. Yen et al. and Nikiforakis et al., as one of the few early studies, discuss and measure the effectiveness of fingerprinting [34, 47]. Acar et al. [11] conduct a large-scale study canvas fingerprinting, evercookies, and the use of "cookie syncing". FPDetective [12] and Fifield el al. [19] both focus on the list of font perspective in browser fingerprinting, e.g., FPDetective performs a measurement study of millions of most popular websites using fonts in the fingerprints. Similarly, Englehardt et al. [18] also conduct a very large-scale study on one million websites about browser fingerprinting, which results in many new features, such as AudioContext. Cao et al. [16] and Boda et al. [14] study a different angle of browser fingerprinting, i.e., cross-browser fingerprinting. Vastel et al. [41] study the inconsistencies in browser fingerprints and shows such inconsistency brings additional entropy for fingerprinting. There are also many works focusing on different perspectives of browser fingerprinting, such as canvas-based [30], JavaScript engine [29, 31], and hardware-based [32]. Particularly, Laperdrix et al. [25] designs a website, called AmIUnique, and conduct a comprehensive study on 17 attributes of browser fingerprinting.

In terms of measurement study, Gómez-Boix et al. [20], similar to our study, deployed a fingerprinting tool on a real-world website and studied the effectiveness of browser fingerprinting. Note that their study adopts cookies as identifiers to differentiate browser instances. However, to the contrary, our study reveals that both users and browsers, such as Safari powered by Intelligent Tracking Preventing, do delete cookies very often and therefore cookies are unreliable in terms of serving as a ground-truth identifier. In addition, their study focuses on the fingerprinting effectiveness but not dynamics.

6.3 Anti-tracking

We also discuss existing anti-tracking from two aspects: defense against cookie-based and anti-fingerprinting.

6.3.1 Defense against Cookie- or Supercookie-based Tracking. ShareMeNot [38] is a browser add-on to defend against social media button tracking, such as Facebook Like button. Private browsing mode [44, 46] creates an isolated browser profile from the normal ones so that the web user's information, such as cookies, are not preserved. Similarly, TrackingFree [35] proposes to isolate user's website visits via an indegree-bounded graph. The Do Not Track (DNT) [43] header, an opt-out approach, allows a user to ask websites not to track. On the other hand, Meng et al. [27] design a client-side policy that empowers users to control whether to be tracked. Intelligent Tracking Prevention [6] is an anti-tracking approach proposed by WebKit to automatically purge out tracking cookies based on an ML-based detector.

6.3.2 Anti-fingerprinting. Tor Browser [36], a privacy-preserving browser, make many fingerprinting features uniform so that they stay the same across browsers. In addition to Tor Browser, which strictly pursue privacy over functionality, some other browsers often provide a privacy-enhancing mode to protect users from browser fingerprinting. For example, Brave Browser [1] provides a fingerprinting protection mode and Firefox provides Tracking Protection in its private browsing mode. In addition to browsers, some browser add-ons, such as Canvas Defender [2], also provide protections against fingerprinting by adding noises. The research community also works on anti-fingerprinting works. PriVaricator [33] adds randomized noise to fingerprinting results so that a tracker cannot obtain an accurate fingerprint. Deterministic Browser [15], is similar to Tor Browser, but mostly focuses on and defends against timing-based fingerprinting. Recently, W3C also introduces a new group note [7] with several suggested practices to browser vendors on the mitigation of browser fingerprinting.

7 CONCLUSION

Browser fingerprints are dynamic, i.e., they evolve over time when users update browsers and OS, or even just interact with their browsers. Such fingerprint dynamics will bring inaccuracies for existing fingerprinting tools to track web users. In this paper, we perform the first large-scale measurement study on the dynamics of browser fingerprints by deploying a customized fingerprinting tool at a real-world website and collecting millions of data over an eight-month period. We then process the collected raw data by generating a dynamics dataset with browser instances represented by browser ID, i.e., a combination of an anonymized version of username provided by the deployment website and some stable browser features.

Our results show that fingerprint dynamics can be classified into three major categories based on their root causes: browser or OS updates, user actions, and environment updates. Our study further yield several new insights: (i) fingerprint dynamics may leak security- or privacy-related information, (ii) prior evolution-aware fingerprinting tools, e.g., FP-Stalker, perform poorly in a large-scale, real-world setting, (iii) some unrelated fingerprint features may be correlated in a piece of dynamics, and (iv) fingerprinting dynamics can be correlated with real-world events like browser or OS updates. We also give several pieces of advices to browser vendors and users on security and privacy as well as evolution-aware fingerprinting tool developers on improving the linking accuracy and speed.

In the future, we believe that it would be interesting to study the trade-off between uniqueness and linkability of browser fingerprints on our large-scale dataset. We would like to design a better fingerprinting tool that balances these two important metrics in browser fingerprinting because uniqueness defines to what extent the tool can track a browser instance and linkability defines how long the tool can track a browser instance.

ACKNOWLEDGEMENT

We would like to thank our shepherd, Tobias Bajwa, and anonymous reviewers for their helpful comments and feedback. This work was supported in part by National Science Foundation (NSF) grant CNS-18-54001 and an Amazon Research Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or Amazon.

REFERENCES

[1] Brave browser. https://brave.com/.
[2] Canvas defender. https://chrome.google.com/webstore/detail/canvas-defender/obdbgnebcljmgkoljcdddaopadkifnpm?hl=en.
[3] Detecting system fonts without flash. https://www.bramstein.com/writing/detecting-system-fonts-without-flash.html.
[4] Device / browser fingerprinting - heuristic-based authentication. https://docs.secureauth.com/pages/viewpage.action?pageId=33063454.
[5] Github repository of our measurement tool. https://github.com/bfMeasurement/bfMeasurement.
[6] Intelligent tracking prevention. https://webkit.org/blog/7675/intelligent-tracking-prevention/.
[7] Mitigating browser fingerprinting in web specifications. https://www.w3.org/TR/fingerprinting-guidance/.
[8] Modern & flexible browser fingerprinting library. https://github.com/Valve/fingerprintjs2.
[9] Online comments on firefox 57. https://www.cnet.com/forums/discussions/firefox-57-is-awful/.
[10] Panopticlick: Is your browser safe against tracking? https://panopticlick.eff.org/.
[11] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), CCS '14, pp. 674–689.
[12] Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. FPDetective: Dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security* (2013), CCS '13, pp. 1129–1140.
[13] Ayenson, M., Wambach, D., Soltani, A., Good, N., and Hoofnagle, C. Flash cookies and privacy ii: Now with html5 and etag respawning. *Available at SSRN 1898390* (2011).
[14] Boda, K., Földes, A. M., Gulyás, G. G., and Imre, S. User tracking on the web via cross-browser fingerprinting. In *Proceedings of the 16th Nordic Conference on Information Security Technology for Applications* (2012), NordSec'11, pp. 31–46.
[15] Cao, Y., Chen, Z., Li, S., and Wu, S. Deterministic browser. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017* (2017), pp. 163–178.
[16] Cao, Y., Li, S., and Wijmans, E. (cross-)browser fingerprinting via os and hardware level features. In *Annual Network and Distributed System Security Symposium* (2017), NDSS.
[17] Eckersley, P. How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium* (2010), Springer, pp. 1–18.
[18] Englehardt, S., and Narayanan, A. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (2016), CCS '16.
[19] Fifield, D., and Egelman, S. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security*. Springer, 2015, pp. 107–124.
[20] Gómez-Boix, A., Laperdrix, P., and Baudry, B. 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, Apr. 2018), pp. 1–10.
[21] Krishnamurthy, B., Naryshkin, K., and Wills, C. Privacy leakage vs. protection measures: the growing disconnect. In *Web 2.0 Security and Privacy Workshop* (2011).
[22] Krishnamurthy, B., and Wills, C. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web* (2009), ACM, pp. 541–550.
[23] Krishnamurthy, B., and Wills, C. E. Generating a privacy footprint on the internet. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (2006), ACM, pp. 65–70.
[24] Krishnamurthy, B., and Wills, C. E. Characterizing privacy in online social networks. In *Proceedings of the first workshop on Online social networks* (2008), ACM, pp. 37–42.
[25] Laperdrix, P., Rudametkin, W., and Baudry, B. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *37th IEEE Symposium on Security and Privacy (S&P 2016)* (2016).
[26] Lerner, A., Simpson, A. K., Kohno, T., and Roesner, F. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016).
[27] Meng, W., Lee, B., Xing, X., and Lee, W. Trackmeornot: Enabling flexible control on web tracking. In *Proceedings of the 25th International Conference on World Wide Web* (2016), WWW '16, pp. 99–109.
[28] Metwalley, H., and Traverso, S. Unsupervised detection of web trackers. In *Globecom* (2015).
[29] Mowery, K., Bogenreif, D., Yilek, S., and Shacham, H. Fingerprinting information in javascript implementations.
[30] Mowery, K., and Shacham, H. Pixel perfect: Fingerprinting canvas in html5.
[31] Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., and Wien, F. Fast and reliable browser identification with javascript engine fingerprinting. In *W2SP* (2013).
[32] Nakibly, G., Shelef, G., and Yudilevich, S. Hardware fingerprinting using html5. *arXiv preprint arXiv:1503.01408* (2015).
[33] Nikiforakis, N., Joosen, W., and Livshits, B. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web* (2015), WWW '15, pp. 820–830.
[34] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., and Vigna, G. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symposium on Security and Privacy* (2013).
[35] Pan, X., Cao, Y., and Chen, Y. I do not know what you visited last summer - protecting users from third-party web tracking with trackingfree browser. In *NDSS* (2015).
[36] Perry, M., Clark, E., and Murdoch, S. The design and implementation of the tor browser [draft][online], united states, 2015.
[37] Pugliese, G., Riess, C., Gassmann, F., and Benenson, Z. Long-term observation on browser fingerprinting: Users' trackability and perspective. *Proceedings on Privacy Enhancing Technologies 2* (2020), 558–577.

[38] Roesner, F., Kohno, T., and Wetherall, D. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (2012), NSDI'12, pp. 12–12.
[39] Soltani, A., Canty, S., Mayo, Q., Thomas, L., and Hoofnagle, C. J. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management* (2010).
[40] Vastel, A., Laperdrix, P., Rudametkin, W., and Rouvoy, R. Fp-stalker: Tracking browser fingerprint evolutions along time. In *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 54–67.
[41] Vastel, A., Laperdrix, P., Rudametkin, W., and Rouvoy, R. Fp-scanner: the privacy implications of browser fingerprint inconsistencies. In *Proceedings of the 27th USENIX Security Symposium* (2018).
[42] Wang, J., Zhang, W., and Yuan, S. *Display Advertising with Real-Time Bidding (RTB) and Behavioural Targeting*. Now Publishers Inc., Hanover, MA, USA, 2017.
[43] Wikipedia. Do Not Track Policy. http://en.wikipedia.org/wiki/Do_Not_Track_Policy.
[44] Wikipedia. Privacy Mode. http://en.wikipedia.org/wiki/Privacy_mode.
[45] Wu, S., Li, S., Cao, Y., and Wang, N. Rendered private: Making GLSL execution uniform to prevent webgl-based browser fingerprinting. In *28th USENIX Security Symposium (USENIX Security 19)* (Santa Clara, CA, Aug. 2019), USENIX Association, pp. 1645–1660.
[46] Xu, M., Jang, Y., Xing, X., Kim, T., and Lee, W. Ucognito: Private browsing without tears. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (2015), CCS '15, pp. 438–449.
[47] Yen, T.-F., Xie, Y., Yu, F., Yu, R. P., and Abadi, M. Host fingerprinting and tracking on the web: Privacy and security implications. In *Proceedings of NDSS* (2012).
[48] Yuan, S., Wang, J., and Zhao, X. Real-time bidding for online advertising: Measurement and analysis. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising* (New York, NY, USA, 2013), ADKDD '13, ACM, pp. 3:1–3:8.

A DETAILED DYNAMICS LEADING TO PRIVACY LEAKS

A.1 Leak of the Installation of Microsoft Office Pro Plus 2013

In this subsection, we present the list of fonts (127 fonts in total) added by the installation of Microsoft Office Pro Plus 2013 or beyond below.

"Arial Unicode MS", "Bodoni MT Condensed", "Stencil", "Perpetua Titling MT", "Haettenschweiler", "Matura MT Script Capitals", "Arial Narrow", "Elephant", "Gill Sans MT Ext Condensed Bold", "Franklin Gothic Demi Cond", "Bodoni MT", "Palace Script MT", "Modern No. 20", "Perpetua", "Wide Latin", "Kunstler Script", "Rockwell Extra Bold", "Bell MT", "Lucida Sans", "Harrington", "MS Reference Sans Serif", "Lucida Handwriting", "Vivaldi", "Gill Sans Ultra Bold", "Bookshelf Symbol 7", "Rage Italic", "Agency FB", "Eras Bold ITC", "Old English Text MT", "Broadway", "Franklin Gothic Book", "Copperplate Gothic Light", "Wingdings 3", "Wingdings 2", "Snap ITC", "Franklin Gothic Demi", "Forte", "Segoe UI Semilight", "Gigi", "Lucida Calligraphy", "Arial Rounded MT Bold", "Colonna MT", "Bauhaus 93", "Poor Richard", "Gill Sans MT", "Centaur", "MS Reference Specialty", "Imprint MT Shadow", "Franklin Gothic Medium Cond", "Copperplate Gothic Bold", "Playbill", "Harlow Solid Italic", "Footlight MT Light", "Calibri Light", "Century Gothic", "Viner Hand ITC", "Bradley Hand ITC", "Calisto MT", "Eras Light ITC", "MS Outlook", "Parchment", "Lucida Sans Typewriter", "Bodoni MT Black", "Engravers MT", "Mistral", "Goudy Stout", "Pristina", "Garamond", "Book Antiqua", "Brush Script MT", "High Tower Text", "Niagara Solid", "Ravie", "Gill Sans MT Condensed", "Lucida Fax", "Informal Roman", "Algerian", "Maiandra GD", "Tw Cen MT Condensed", "Edwardian Script ITC", "Britannic Bold", "OCR A Extended", "Bodoni MT Poster Compressed", "Tempus Sans ITC", "Eras Demi ITC", "Jokerman", "Century", "Niagara Engraved", "Magneto", "French Script MT", "Tw Cen MT", "Lucida Bright", "Tw Cen MT Condensed Extra Bold", "Gadugi", "Rockwell Condensed", "Castellar", "Script MT Bold", "Microsoft YaHei UI", "Freestyle Script", "Blackadder ITC", "Gloucester MT Extra Condensed", "Bernard MT Condensed", "Curlz MT", "Felix Titling", "Baskerville Old Face", "Vladimir Script", "Rockwell", "Monotype Corsiva", "Onyx", "Kristen ITC", "Franklin Gothic Heavy", "Cooper Black", "Bookman Old Style", "Eras Medium ITC", "Californian FB", "Goudy Old Style", "Gill Sans Ultra Bold Condensed", "Papyrus", "Chiller", "Berlin Sans FB Demi", "Showcard Gothic", "Juice ITC", "Nirmala UI", "Berlin Sans FB", "Microsoft JhengHei UI", "Century Schoolbook", "MT Extra".

Note that because not all the fonts are collected by our fingerprinting tool, we use the following list (96 fonts in total) to detect the installation of Microsoft Office.

"Bodoni MT Condensed", "Stencil", "Perpetua Titling MT", "Haettenschweiler", "Matura MT Script Capitals", "Elephant", "Gill Sans MT Ext Condensed Bold", "Palace Script MT", "Modern No. 20", "Perpetua", "Wide Latin", "Kunstler Script", "Rockwell Extra Bold", "Bell MT", "Harrington", "Vivaldi", "Gill Sans Ultra Bold", "Bookshelf Symbol 7", "Rage Italic", "Agency FB", "Eras Bold ITC", "Old English Text MT", "Broadway", "Copperplate Gothic Light", "Snap ITC", "Forte", "Gigi", "Rockwell Condensed", "Colonna MT", "Bauhaus 93", "Poor Richard", "Gill Sans MT", "Centaur", "MS Reference Specialty", "Imprint MT Shadow", "Copperplate Gothic Bold", "Playbill", "Harlow Solid Italic", "Footlight MT Light", "Viner Hand ITC", "Bradley Hand ITC", "Calisto MT", "Eras Light ITC", "Parchment", "Bodoni MT Black", "Engravers MT", "Mistral", "Goudy Stout", "Pristina", "Brush Script MT", "High Tower Text", "Niagara Solid", "Ravie", "Gill Sans MT Condensed", "Informal Roman", "Algerian", "Maiandra GD", "Tw Cen MT Condensed", "Edwardian Script ITC", "Britannic Bold", "OCR A Extended", "Bodoni MT Poster Compressed", "Tempus Sans ITC", "Eras Demi ITC", "Jokerman", "Niagara Engraved", "Magneto", "French Script MT", "Tw Cen MT", "Berlin Sans FB Demi", "Tw Cen MT Condensed Extra Bold", "Castellar", "Script MT Bold", "Freestyle Script", "Blackadder ITC", "Gloucester MT Extra Condensed", "Bernard MT Condensed", "Curlz MT", "Felix Titling", "Baskerville Old Face", "Vladimir Script", "Rockwell", "Onyx", "Kristen ITC", "Bodoni MT", "Cooper Black", "Eras Medium ITC", "Californian FB", "Goudy Old Style", "Gill Sans Ultra Bold Condensed", "Papyrus", "Chiller", "Showcard Gothic", "Juice ITC", "Berlin Sans FB", "MT Extra".

A.2 Leak of One Windows 7 Update Released on April 22, 2014

The Windows 7 update will change canvas rendering hash value from Value One: 14578bcaee87ff6fe7fee38ddfa2306a7e3b0a0a to Value Two: bd554a7d5da9293cf3fed52d2052b2b948a14b77.

A.3 Leak of LibreOffice6 Installation

In this subsection, we present the list of fonts added by the installation of LibreOffice6 below.

"Miriam Mono CLM", "Noto Sans Lisu", "Scheherazade", "Linux Libertine Display G", "EmojiOne Color", "Noto Naskh Arabic", "Linux Biolinum G", "Source Code Pro Black", "Noto Sans Light", "Frank Ruehl CLM", "Caladea", "Noto Serif", "OpenSymbol", "Rubik", "Noto Sans Georgian", "Noto Sans Lao", "Liberation Sans", "Source Code Pro Light", "Noto Serif Lao", "DejaVu Serif Condensed", "KacstBook", "DejaVu Sans Light", "Reem Kufi Regular", "Source Code Pro Semibold", "Noto Naskh Arabic UI", "Source Sans Pro Black", "Gentium Basic", "DejaVu Math TeX Gyre", "Source Code Pro ExtraLight", "Noto Kufi Arabic", "Noto Sans Hebrew", "Amiri", "Source Sans Pro Semibold", "Miriam CLM", "Source Code Pro", "Source Sans Pro", "Noto Sans Cond", "Liberation Serif", "KacstOffice", "Source Code Pro Medium", "DejaVu Sans", "Liberation Mono", "Noto Serif Armenian", "Alef", "Gentium Book Basic", "David Libre", "Noto Sans Armenian", "Noto Serif Cond", "Linux Libertine G", "Liberation Sans Narrow", "DejaVu Sans Condensed", "Source Sans Pro ExtraLight", "DejaVu Sans Mono", "Noto Sans Arabic UI", "Noto Serif Georgian", "Noto Mono", "David CLM", "Carlito", "Amiri Quran", "DejaVu Serif", "Noto Serif Hebrew", "Noto Serif Light", "Source Sans Pro Light", "Noto Sans", "Noto Sans Arabic".

A.4 Fonts Added by Firefox 57

In this subsection, we list all the fonts added by Firefox 57 below.

"Arial Black", "Arial Narrow", "Arial Rounded MT Bold", "Segoe UI Light", "Segoe UI Semibold", "Berlin Sans FB Demi", "Bernard MT Condensed", "Bodoni MT Black", "Bodoni MT Condensed", "Bodoni MT Poster Compressed", "Britannic Bold", "Cooper Black", "Copperplate Gothic Bold", "Copperplate Gothic Light", "Footlight MT

Light", "Gill Sans MT Condensed", "Gill Sans MT Ext Condensed Bold", "Gill Sans Ultra Bold", "Gill Sans Ultra Bold Condensed", "Harlow Solid Italic", "OCR A Extended", "Rage Italic", "Rockwell Condensed", "Rockwell Extra Bold", "Script MT Bold", "Tw Cen MT Condensed", "Tw Cen MT Condensed Extra Bold".