

Eadem, Sed Aliter: Investigating the Instability of Browser Fingerprinting

Xu Lin
Washington State University
xu.lin@wsu.edu

Shuijiang Wu
F5, Inc.
buaasniper@gmail.com

Jacob Anthony Eckfeldt
Washington State University
jacob.eckfeldt@wsu.edu

Pengfei Sun
F5, Inc.
p.sun@f5.com

Jason Polakis
University of Illinois Chicago
polakis@uic.edu

ABSTRACT

The use of browser fingerprinting has become prevalent in modern web services, both as part of risk-based authentication mechanisms as well as for tracking purposes. In this paper, we present an in-depth investigation of a large dataset of browser fingerprints from users authenticating with high-value web services, procured by F5, Inc. Our analysis results in novel observations and insights that subvert our community’s established understanding of browser fingerprinting, by revealing the inherent *instability* of prevalent attributes even within stable execution environments. In other words, we identify patterns wherein fingerprints change *even when no change has been made to the device* (e.g., due to minor deviations in browsers’ floating point arithmetics or HTML element-size calculations). Naturally, this poses a major obstacle to the real-world effectiveness of browser-fingerprinting. To that end, we propose Predictive-FP, a novel adaptive fingerprinting strategy that leverages historical population-wide information to anticipate unstable fingerprinting values while also ignoring ephemeral attributes (e.g., version numbers). Our extensive experimental evaluation demonstrates how our system significantly outperforms the state of the art, especially over longer periods of time, and is able to re-identify ~51% more devices after 6 months. While our technique enables more robust authentication, it also presents a potential privacy risk. Accordingly, we develop a configurable privacy-preserving extension for users, that strategically modifies an extensive set of fingerprinting attributes to prevent tracking on non-allowlisted sites.

CCS CONCEPTS

• **Security and privacy** → **Browser security; Privacy protections; Authentication.**

KEYWORDS

Measurement; Browser Fingerprinting; Tracking; Risk-based Authentication

ACM Reference Format:

Xu Lin, Shuijiang Wu, Jacob Anthony Eckfeldt, Pengfei Sun, and Jason Polakis. 2026. Eadem, Sed Aliter: Investigating the Instability of Browser

Fingerprinting. In *Proceedings of the 2026 ACM Internet Measurement Conference (IMC '26)*, October 12–16, 2026, Karlsruhe, Germany. ACM, New York, NY, USA, 18 pages. <https://doi.org/10.1145/3777912.3809154>

1 INTRODUCTION

Over the last decade, browser fingerprinting has garnered significant attention from researchers, while also being widely deployed in practice. In essence, browser fingerprinting techniques aim to uniquely identify user devices based on a collection of *attributes* that describe aspects of the device’s hardware components (e.g., GPU vendor) and software stack (e.g., operating system), or *objects* that may implicitly contain “artifacts” that are the byproducts of the user’s computing environment during the generation process (e.g., different browser and GPU combinations will render the same WebGL element differently). Each fingerprinting attribute or object exhibits an amount of entropy, based on its uniqueness within the user population. Web services try to maximize the aggregate amount of entropy by leveraging a rich and diverse set of fingerprinting techniques and signals.

Prior work has extensively explored different fingerprinting techniques, the most common being the use of different JavaScript APIs for obtaining system characteristics [16, 28, 37]. While browser fingerprinting techniques provide useful and powerful capabilities, they also suffer from a known limitation: fingerprinting attributes change over time due to changes that occur to the user’s device. Vastel et al. [57] found that fingerprints tend to change frequently due to *software updates* or changes to the system’s configuration.

In this paper, we conduct an IRB-exempt in-depth investigation of a large-scale fingerprinting dataset from F5, Inc., which contains fingerprinting data from the authentication flows of over *seven million* users. This dataset provides a behind-the-scenes real-world view of fingerprint collection and use in the authentication workflow of five high-value web services. Our data-driven analysis uncovers previously-unknown problematic characteristics of browser fingerprints, which present major pitfalls in the deployment of fingerprinting systems in practice.

First, we find that prevalent fingerprinting attributes are *unstable even when the user’s computing environment has not changed*, wherein subsequent sessions from users with the same browser will exhibit different corresponding fingerprints. We then experimentally verify that multiple fingerprinting signals can change within the same execution environment. In more detail, our experiments demonstrate that techniques that provide high-entropy fingerprinting attributes (e.g., canvas elements, list of fonts) can,



This work is licensed under a Creative Commons Attribution 4.0 International License. *IMC '26, October 12–16, 2026, Karlsruhe, Germany*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2327-8/2026/10
<https://doi.org/10.1145/3777912.3809154>

indeed, produce different results *without any changes to the user's device*. Second, we find that certain fingerprinting techniques are particularly problematic for specific browser and operating system combinations, which we refer to as *discordant-combos*. Specifically, our large scale measurement and experimental verification reveal that certain techniques exhibit considerably higher instability in certain execution environments. For instance, font fingerprinting techniques are particularly unstable for Windows devices running the Firefox browser. Similarly, the WebGL renderer attribute is unstable when the user is browsing over Chrome on a Linux-based machine. Finally, we observe a strange behavior wherein the unstable fingerprinting values will often flip-flop between specific combinations of values. These *FP-flips* are of a varied nature, ranging from binary attributes flipping (e.g., a specific font family being found only in certain sessions) to more obscure changes (e.g., the WebGL Vendor switching between Google Inc. (Intel Inc.) and Google Inc. (Google)). Our root cause analysis reveals how subtle differences in browser calculations (e.g., when measuring the size of the dock in MacOS, or fractional values being rounded during certain Canvas rendering operations) result in instability.

Driven by these findings, we develop a novel adaptive fingerprinting strategy designed to overcome the challenges introduced by the unstable nature of fingerprints. Predictive-FP essentially anticipates a user's future unstable fingerprinting values based on population-wide observations, while ignoring ephemeral attributes (e.g., version numbers). Despite the inherent instability and evolution of fingerprints, most future variations are likely to fall within a set anticipated by our system. Our approach ensures accurate long-term identification of specific devices while maintaining high discriminative power, thus enabling more robust authentication systems. We conduct an extensive evaluation of Predictive-FP on our real-world dataset of millions of users, and demonstrate that our system greatly outperforms existing approaches and successfully overcomes the obstacles presented by fingerprint instability and FP-flips. This includes using Predictive-FP to re-identify almost 7 million recurring users of a major financial institution across a 6-month period. Predictive-FP generated consistent identifiers for 94.37% of users, significantly surpassing the 43.44% achieved by the state of the art [57]. Due to the potential privacy risk of our technique (i.e., being used for tracking instead of augmenting risk-based authentication systems), we develop a browser extension that randomizes an extensive selection of fingerprinting signals. Importantly, it allows users to selectively allowlist specific domains, thereby reaping the benefits of augmented risk-based authentication on high-value sites while preventing tracking on other sites.

In summary, our work has the following contributions:

- We conduct an in-depth, large-scale analysis of prevalent browser fingerprinting techniques, and provide novel insights on their inherent limitations.
- We develop *Predictive-FP*, a novel adaptive fingerprinting strategy that overcomes the challenges posed by the unstable nature of browser fingerprints. We have open sourced our code to facilitate researchers and practitioners [1].
- We experimentally demonstrate the capabilities of our system, which significantly outperforms the state of the art, both in discriminative power and long-term effectiveness.

Table 1: Browser fingerprint dataset. Cookies[†] denotes the number of cookies of recurring users in 08/23. LoginID[†] and LoginID* represents the recurring users in 08/23 and between 09/23 - 02/24, respectively.

Company	Sessions	Cookies [†]	LoginID [†]	LoginID*
Bank-A	169,219,670	11,342,072	7,449,311	6,838,475
Bank-B	8,086,261	112,004	104,749	163,391
Credit-A	6,080,950	102,637	64,670	59,386
Retail-A	449,410	64,479	62,322	17,170
Bank-C	192,984	1,570	1,493	4,340

- We develop an extensive fingerprint-randomizing extension as a mitigation against privacy-invasive sites [1].

2 BROWSER FINGERPRINT INSTABILITY

Here, we present an initial measurement of fingerprinting instability; more in-depth explorations and insights are provided in the following subsections.

Data collection. Browser fingerprinting uses diverse techniques for collecting or inferring information about the user's device. In practice, web services typically leverage HTTP headers and JavaScript APIs. For our large-scale analysis, we use a dataset procured by F5, Inc. which includes the browser fingerprints and anonymized metadata (e.g., cookie value) extracted from the HTTP POST requests sent during the login process of five major web services. Each successful login attempt is associated with one session; the fingerprinting process occurs once at login time, and is not used as part of a continuous risk-based authentication analysis throughout the session. Moreover, the dataset was derived from sessions where the users were flagged as benign by the anti-bot and anti-fraud security solutions deployed by F5, Inc., and users were able to successfully complete the authentication workflow (which may include two-factor authentication challenges). The anti-bot and anti-fraud systems are learning-based, and incorporate user feedback and expert manual review to ensure labelling accuracy.

Dataset breakdown and usage. Logins occurred between August 2023 and February 2024, as shown in Table 1. The table offers an overview of the number of cookies, user accounts, and average sessions per cookie for each company. To ensure user privacy, we differentiate user accounts through LoginID, a unique anonymized identifier that is generated by F5, Inc. (all data was anonymized by F5, Inc. before being shared, including all identifiers of users and F5's customers.) For clarity, next we detail the roles of each dataset in our analysis and certain terms used in our analysis. We use August 2023 as a self-contained one-month window to characterize fingerprint instability, isolating inherent variability from long-term software evolution. The full six-month period (September 2023–February 2024) is used exclusively for evaluating Predictive-FP (§4). We primarily focus on Bank-A due to its scale (>169M sessions and >7M users), while the remaining four companies serve for cross-validation to confirm that the observed trends generalize. The criteria for *returning users* depends on the analysis window. For August, we require at least two sessions within the month to capture within-month instability, whereas for the six-month period, we require at least one session in both September and February to measure re-identification across the widest possible temporal

Table 2: Information included in our dataset (metadata and browser fingerprints) along with example values.

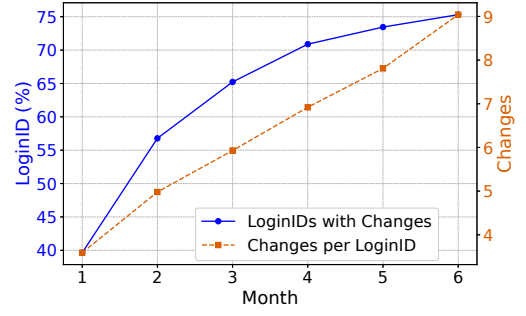
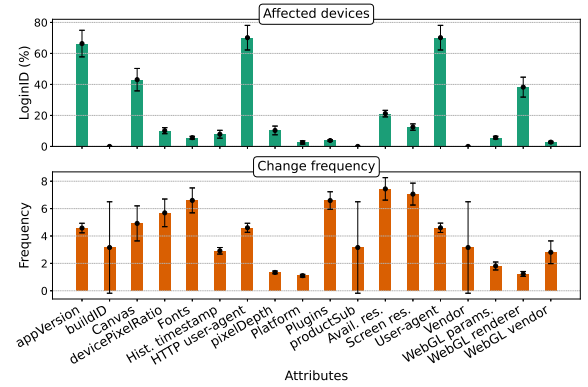
Metadata	
LoginID	Hash value
Cookie	Hash value
session timestamp	2024-02-08 21:41:16.289000+00:00
Browser Fingerprint Attributes: Examples and Unique Value Counts	
appVersion	5.0 (WindowsNT10.0; Win64;x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0Safari/537.36 316,610
buildID	20181001000000 171
Canvas	Hash value 3,373,520
Historical timestamp	Hash value 2,182
devicePixelRatio	1.25 13,526
HTTP user-agent	Mozilla/5.0 (WindowsNT10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0Safari/537.36 476,390
Fonts	Hash value 49,500
Vendor	Apple Computer, Inc. 9
pixelDepth	30 8
Platform	Linux armv8l 37
Plugins	Hash value 1,213,524
productSub	20030107 3
Available resolution	1536x824 348,364
Screen resolution	1440x900 295,001
User-agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 517,795
WebGL parameters	Hash value 20,400
WebGL renderer	Mesa DRI Intel(R) UHD Graphics 600 (GLK 2) 24,171
WebGL vendor	Intel Open Source Technology Center 7,224

gap. Since data is obtained over multiple months, a single user may log in from multiple devices; we consider only the cookie with the highest number of session records for each user, unless stated otherwise. As a result, the *LoginID*⁺ and *LoginID*^{*} counts in Table 1 are not directly comparable, as they reflect different selection criteria rather than user attrition.

Table 2 lists the browser fingerprint attributes that we use, which are typical for both commercial and academic tools. In each case, the attributes are associated with a LoginID, cookie values, and the session timestamp. We note that certain attributes are stored as hash values – this is common practice, especially for signals that are vectors of pre-determined elements (e.g., supported fonts) or values that are too large (e.g., canvas uses a hash to capture a device’s unique rendering output without storing the full image). The two user-agent attributes differ in how they are collected: HTTP user-agent comes from the HTTP request header, while user-agent is obtained via navigator.userAgent. Their cardinalities differ because certain tokens (e.g., Mobile) may appear in one but not the other, as we discuss in §2.2.

2.1 Unstable Software Combinations

Attribute changes. We find that $\approx 40\%$ of devices experience changes in the first month, increasing steadily to over 75% by month 6 (see Figure 1). Similarly, the number of changes per device begins at 3.6 and rises to 9 by month 6. Figure 2 illustrates the attribute instabilities observed across four companies for which we have comprehensive six-month data. Attributes such as appVersion and user-agent experience changes in $\approx 70\%$ of user devices, as expected. Interestingly, canvas and WebGL renderer exhibit changes in about 40% of devices. In terms of change frequency, screen available resolution (hereafter referred to as available resolution) has the highest rate, with over seven changes per device, whereas platform has the lowest, with slightly over one change per

**Figure 1: Percentage of LoginIDs with fingerprint changes and number of changes per LoginID (09/2023–02/2024).****Figure 2: Attribute instability over six months. LoginID refers to IDs that experienced changes, and Frequency denotes the avg number of changes observed.**

device. Regarding change intervals, pixelDepth demonstrates the longest duration between changes (63 days), while attributes such as fonts, plugins, and resolution experience shorter intervals of around 14 days, indicating more frequent changes.

While, as one would expect, changes to the device software and hardware lead to attribute changes, we find that attribute values can change even with a steady device configuration. Next, we describe our novel insights on the instability of fingerprinting techniques, based on our observations from tens of millions of sessions from millions of users. We detail the behaviors and patterns we observed in the wild and experimentally investigate their root causes by reproducing them in a controlled setting (§2.2, §2.3, §2.4, §2.5).

2.2 Instability for Unchanged Browsers

We performed a data-driven analysis of the browser fingerprints collected from benign users during the authentication workflow when connecting to the five web services. Our initial focus of exploring potential patterns in how users’ fingerprints change over time (e.g., due to updating their system) resulted in uncovering an unexpected finding. Specifically, our experiments revealed a considerable number of instances where specific fingerprinting attributes changed despite the browser version remaining the same. More importantly, several of these attributes are either generally accepted

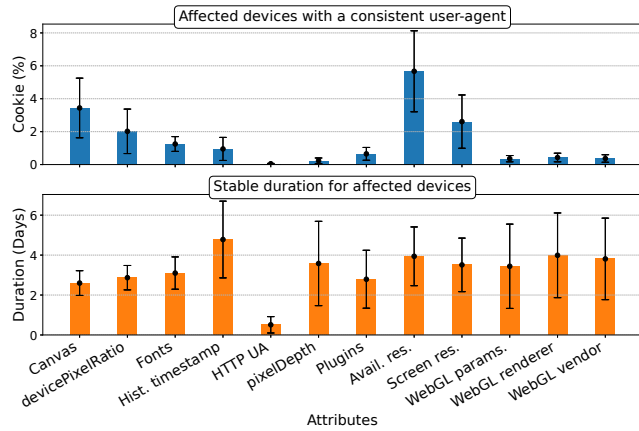


Figure 3: Unstable attributes with an unchanged user-agent (August), and duration of time during which an attribute’s value remained unchanged.

as invariant unless the user changes devices (e.g., a mobile device’s resolution), or at least stable until a major hardware upgrade is made to the device (e.g., the WebGL vendor).

To further explore this phenomenon, for each device we identified the fingerprinting attribute values that differ across different sessions while the user-agent remains consistent. In other words, we identify potential inconsistencies that may occur in specific fingerprinting attributes without being accompanied by a change in the user-agent (which could justify such changes due to differences in the user’s browser). We note that an unchanged user-agent is a necessary but not sufficient condition for ruling out software updates, since Chrome’s user-agent reduction [55] suppresses minor version changes. As such, wherever we state that “no browser update”, that does not preclude minor version changes having occurred. Also, we note that by focusing on unique cookie values, and not LoginIDs, we are focusing our analysis on specific user devices and not “polluting” our analysis by including cases where users connect to their accounts from multiple devices (which we treat as separate cases).

Figure 3 presents aggregate statistics for attributes across the five companies when the user-agent remains unchanged. In general, resolution-related attributes are by far the most unstable features. In particular, available resolution exhibits the highest instability across all companies, reaching 9.10% at Bank-A and remaining above 3% in every company. Screen resolution also shows consistently high instability rates, while canvas reaches 5.88% at Bank-B. Attributes like devicePixelRatio (2.02%) and fonts (1.25%) show moderate instability, while WebGL-based attributes remain largely stable, with instability rates below 1% across all companies.

We then focus on devices with a consistent user-agent that nonetheless exhibit instability within a one-month period. Most attributes exhibit similar stability, typically remaining unchanged for several days before a variation occurs. For example, available resolution remains stable for an average of 3.9 days across companies, while canvas remains stable for about 2.6 days. Even attributes with low instability rates show comparable stability duration, such as the 3.4 days observed for WebGL parameters. Across all attributes, the average stable duration is under 8 days, and most

remain stable for fewer than 4 days. We also examined whether companies with more sessions per device or larger datasets tend to have shorter stability periods, but the data does not support this. For example, Credit-A, with over 102K cookies and the highest average sessions per cookie (9.18), shows a longer stability period for canvas (3d 13h), while Retail-A, with 64K cookies and fewer sessions per cookie (3.39), exhibits a shorter period (2d 18h).

Our findings highlight the growing instability of fingerprinting attributes across different companies, questioning their reliability for consistent and effective device identification. These results demonstrate that properties often assumed to be stable, such as display size and rendering outputs, can frequently fluctuate in real-world settings. Moreover, these variations suggest that while certain attributes are generally prone to instability, the extent and nature of this instability can differ significantly depending on other factors not mandated by the company’s size or user base.

Root cause analysis of unstable resolution. When examining the most unstable attribute, available resolution, we find that the six most common changes constitute significant value changes, such as from 1536×824 to 1920×1040 , indicating users likely changed their display or resolution settings. Interestingly, the next five most common changes involve only minor variations in availHeight, such as from 807 to 809, and these changes are not attributed to software updates. They are observed on Mac devices, and the majority of them occur without any browser updates (>90%). For example, 92.66% of those devices experienced a change from 1440×809 to 1440×813 . The attribute value represents the usable screen space after subtracting elements such as the top menu bar and dock. To further investigate this issue, we modified the source code of Chromium Version 132.0.6799.0 on macOS to log the availHeight. We find that such variations occur due to the browser computing different dock heights. On our device, the dock height varied among 70, 68, and 67 pixels after restarting the browser, resulting in availHeight values of 874, 876, or 877 pixels. In some cases, we observed the dock size shifting from 70 to 68 or from 68 to 67 pixels within the same session; these changes occurred without us changing the dock or browser window.

Unexpected instability in related attributes. An additional finding involves instability in related fingerprinting attributes, which are expected to remain unchanged when the user-agent has not changed. Excluding the relatively small dataset from Bank-C, we find that across the other four companies, 0.05% - 0.07% of devices exhibit an unstable HTTP user-agent. Our analysis revealed that one cause for this is that, on Android, the Chrome browser *sometimes* adds the token Mobile to the user-agent strings, and this token may be present in either the HTTP header or the value reported through JavaScript. For example, the user-agent string Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Mobile Safari/537.36 may appear without the Mobile token in some sessions. A similar behavior occurs with the token Edg in some sessions of the Edge browser on Windows.

Surprisingly, we also find instances of platform being unstable in three of the services, albeit at a smaller scale (0.002% - 0.03%). In Bank-A, the two most frequent changes are between Linux armv81 and Linux aarch64, occurring in both directions and affecting 1,494 devices. Interestingly, out of these devices, we find 1,344

Table 3: Canvas images used in our experiment.

#	Image	#	Image
1	Xa9!~r&T(4qL@z*	2	Xa9!~r&T(4qL@z*, 😊
3	Jived fox nymph grabs quick waltz, 😊 Xa9!~r&T(4qL@z*, 😊	4	Jived fox nymph grabs quick waltz, 😊 Xa9!~r&T(4qL@z*, 😊
5		6	

showing an iOS device’s user-agent, and five showing a Windows device. The inconsistency between user-agent and platform suggests that these users spoofed one of the attributes’ values (e.g., through a browser extension) or logged in using the embedded browser in a custom app (e.g., FDCasino, My Verizon). For example, a device with the user-agent: Mozilla/5.0 (iPhone; CPU iPhone OS 16_6_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148; GamingWrapper (FDCasino; iOS; iPhone 13,2; 2.57.1) reported a Linux aarch64 platform. Additionally, we find 283 devices where platform changed from iPhone to Win32 on 282 iOS devices and one Android device, again indicating fingerprinting values being modified.

Insight 1: FP-instability. Common fingerprinting attributes that are considered stable in the absence of environmental changes, exhibit instability without a browser upgrade occurring.

2.3 Unstable Software Combinations

During our analysis, we found that certain browsers and platforms exhibit higher instability rates for specific fingerprinting attributes. We break down the most problematic browser and platform combinations for various fingerprinting attributes in Table 7 (see Appendix 2.1). Canvas instability is overwhelmingly concentrated in Samsung Internet browsers on Linux-based mobile platforms, with instability rates exceeding 85% in some cases, far higher than any other browser-platform combination. Available resolution instability is most pronounced on MacIntel platforms across all browsers, typically manifesting as minor pixel-level variations rather than full display changes. Font fingerprinting is particularly unreliable on Win32 Firefox and Linux x86_64 Chrome. WebGL attributes, while generally more stable, show elevated instability on Linux x86_64 Chrome. Historical timestamp instability is particularly notable on mobile devices.

Root cause analysis of unstable canvas. For the canvas attribute, the top four most frequent changes were observed predominantly on Android devices, with nearly 90% of these changes occurring without browser updates. To investigate this instability, we conducted controlled tests on a Samsung Galaxy A15 using the Samsung Internet browser (version 27, based on Chromium 125). The Android device is setup to automatically visit a test site that presents six different canvas images, shown in Table 3 (1-6). These images were designed with increasing levels of complexity: starting with simple black text, then adding emojis and colored text, followed by gradients and shadows, and finally combining curves, colors, gradients, emojis, and text. We visited the testing page ten times and observed that across all ten sessions, the six

images consistently exhibited variations in their canvas values. The source of these differences lies in minor deviations in the color channels (red, green, blue, and alpha) of two pixels that vary from session to session. For example, across two consecutive session in Image 2, one pixel changed from (0, 0, 0, 0) to (0, 2, 0, 0), while another shifted from (0, 3, 0, 0) to (0, 0, 0, 0). Similarly, in Image 6 during the same sessions, one pixel changed from (0, 1, 0, 255) to (0, 0, 0, 255), and another from (0, 0, 0, 0) to (0, 0, 2, 0). Interestingly, the specific pixels that exhibited differences varied across sessions, contributing to the overall instability. This behavior is intentional: Samsung Internet deliberately introduces such variability as part of its anti-tracking feature (we further explore anti-fingerprinting defenses in §2.5).

Unexpected changes in WebGL renderer. Prior studies have regarded this attribute as relatively stable, with only minor variations expected. For instance, in [57] the authors reported that different WebGL renderer values on the same device must have a similarity ratio greater than 0.75, as calculated using the `difflib.SequenceMatcher().ratio` Python function, to account for changes that occur due to software updates. Interestingly, in our analysis of Bank-A’s data, we observed that the two most common value changes had low similarity ratios. The first change, with a ratio of 0.58, involved 8,400 devices, and was mostly observed on Chrome browsers on Chrome OS. The second change, with a ratio of 0.23, was observed in 6,473 devices, primarily running Windows Chrome and Edge.

Insight 2: Discordant-Combos. Certain fingerprinting attributes are particularly unstable in specific combinations of execution environments.

2.4 Patterns within Instability

The next phase of our in-depth data-driven analysis was a closer examination of the values returned by the various unstable fingerprinting attributes. Surprisingly, we find that fingerprinting instability often does not manifest through random value changes, but tends to switch between specific sets of values. Even more surprising is the fact that these changes are not permanent, but instead we observe attributes flip-flopping between values. For example, a specific device may return a canvas element with a hash value of ABC during the first session, then return an element with a hash value of DEF for the *identical* canvas object during the second session, and then flip back to ABC in a future session. As such, despite the instability of fingerprinting attributes, we find that their values tend to switch between a specific set of values. The high percentages of the top FP-flip values across different companies indicate that even though multiple unique changes may occur, a few dominant values usually emerge, showing a predictable pattern in these changes.

For our analysis, we use value pairs to describe the FP-flip behavior in fingerprinting attribute values between sessions. The initial value is the left-side value of the pair, and if the value changes on the following session, the new value is added to the pair as the right-side value, e.g., {ABC, DEF}.

Table 4 summarizes the most common value pair observed for each attribute in the Bank-A dataset (August) and how often these values later reverted. Overall, seven out of 16 attributes exhibited

Table 4: Top pairs for Bank-A (August). Count denotes LoginIDs that experienced this change, and Flip% captures those that reverted to the original value within the next six months. We omit substrings for brevity.

Attribute	Top Pair (comma-separated)	Count	Flip%
User-agent	Mozilla/5.0 (Win...Chrome/115.0..., Mozilla/5.0 (Win...Chrome/116.0...	937,189	0.28
HTTP user-agent	Same as user-agent	937,333	0.32
appVersion	5.0 (Windows NT...Chrome/115.0..., 5.0 (Windows NT...Chrome/116.0...	939,006	0.28
Screen res.	1536 × 864, 1920 × 1080	39,865	84.33
Avail. res.	1536 × 824, 1920 × 1040	22,373	84.79
Canvas	Hash1, Hash2	80,771	89.66
devicePixelRatio	1.25, 1.0	69,887	86.25
Plugins	Hash1, Hash2	36,925	83.83
Hist. timestamp	Hash1, Hash2	27,428	82.36
pixelDepth	30, 24	141,383	42.52
Fonts	Hash1, Hash2	23,471	86.08
Vendor	Apple Computer Inc., Google Inc.	2,143	0
WebGL renderer	Mesa DRI Intel(R) UHD Graphics..., ANGLE (Intel ...Mesa DRI Intel(R)...	8,400	2.87
WebGL params.	Hash1, Hash2	16,849	71.21
WebGL vendor	Intel Open Source Technology Cen..., Google Inc. (Intel Open Source Tech...	14,723	6.59
Platform	Linux armv8l, Linux armv8l	7,000	4.00

a high flipping rate (greater than 80%). Some attributes, such as user-agent, HTTP user-agent, and appVersion, showed very low flipping rates (around 0.3%), because these changes are driven by software updates (e.g., Chrome 115 to 116). The change in navigator.vendor from Apple to Google reflects the evolution of Chromium-based browsers. In contrast, attributes such as canvas and fonts, showed high flipping rates, with over 80% of LoginIDs experienced a FP-flip later. These fluctuations are inherent instabilities that are not caused by software updates or configurations changes. Resolution-related attributes also exhibited notable flipping, suggesting that users or environments switch between these settings and back. These results indicate many rendering- and configurations-related attributes fluctuate over time and often revert.

Root cause of FP-flips in Firefox font fingerprinting. We conducted a controlled experiment in Firefox (versions 139 to 141) on Windows 11 by visiting our test site that used the FingerprintJS code. The experiment tested all 1,044 fonts used by Amiunique [6] and 46 fonts used by our real-world dataset. We observe that Windows Firefox exhibits inconsistent font fallback behavior that can lead to flipping results. Common fingerprinting libraries (e.g., FingerprintJS) use the standard fallback fonts monospace, sans-serif, and serif, and determine a font to be “available” if its rendered dimensions differ from any of these fallback fonts. However, Firefox *sometimes* performs internal font substitutions instead of applying the fallback fonts specified in the code, which disrupts this detection logic and causes unexpected results. These substitutions are defined in the Firefox source [38]. For example, Helvetica is substituted with Arial and both Times and Times New Roman Baltic are resolved to Times New Roman. In some sessions, Firefox detects these fonts as available due to internal substitutions; in others, it treats them as unknown and applies the fallback fonts. We also observe that the fonts Arial Black, Calibri Light, Segoe UI Semilight,

Segoe UI Light, and Sitka are inconsistently detected, despite being available in the system.

To investigate potential causes, we consider that Windows supports two font subsystems: GDI and DirectWrite. We configure Firefox to consistently use DirectWrite and still observe this instability. More specifically, these fonts frequently fail to appear during the first page load following a Firefox restart. Our measurements suggest that this behavior results from a delay in font readiness. On a test device (HP Laptop 14-dk1xxx), these fonts only become available more than one minute after the browser started. To further examine this issue, we add an HTML element styled with one of the unstable fonts. We find that once the styled text is rendered, all five previously unstable fonts become consistently detectable. This suggests that Firefox does not make certain fonts available to JavaScript until at least one of them has been rendered or sufficient time has passed from startup. For instance, after restarting Firefox, text styled with Arial Black using HTML initially renders with a fallback font and only switches to the correct font after a delay of approximately 500 ms. If font fingerprinting occurs before this rendering completes or too soon after Firefox’s startup, it can lead to incorrect results.

Pair concentration in instabilities. We also find that changes in most attributes remain highly concentrated, with only a small fraction of value pairs needed to account for a large portion of the observed variations. Figure 4 presents an analysis of the percentage of unique value pairs required to cover 30%, 50%, and 90% of fingerprinting instabilities observed at Bank-A. The user-agent attribute exhibits the largest number of changes, with 3,676,137 changes across 92,183 unique pairs. Notably, only 2 pairs (0.00%) account for 30% of these changes, 5 pairs (0.01%) account for 50%, and just 82 pairs (0.09%) account for 90%. Another attribute, fonts, requires 5 pairs (0.03%) to cover 30%, 20 pairs (0.13%) to cover 50%, and 1,297 pairs (8.42%) to cover 90%. In contrast, the canvas attribute shows a much more diverse set of FP-flips: while only 20 pairs (0.00%) are needed to cover 30% of changes, it takes 174k pairs (21.17%) to reach 50% and 694k pairs (84.23%) to cover 90%. A similar trend is observed in plugins. We exclude the platform, vendor, pixelDepth, and buildID attributes from this analysis as they are relatively stable and have less than 100 unique pairs each. For 30% coverage, all 13 attributes need less than 0.3% of their pairs; for 50% coverage, 11 out of 13 attributes still remain below 1%. For 90% coverage, seven attributes stay below 10%. Overall, our analysis shows that a small number of value pairs account for the majority of changes that are observed due to the instability of fingerprinting techniques.

Insight 3: FP-Flips. Despite the instability of certain fingerprinting attributes, their values tend to flip-flop between a small set of value pair combinations.

2.5 Anti-fingerprinting Mechanisms

A small number of the instabilities that we observed can be traced back to anti-fingerprinting defenses, which rely on returning randomized values. While major browsers like Chrome and Safari (which account for $\approx 84\%$ of users [53]) do not employ such defenses, more privacy-oriented browsers like Brave and Firefox do

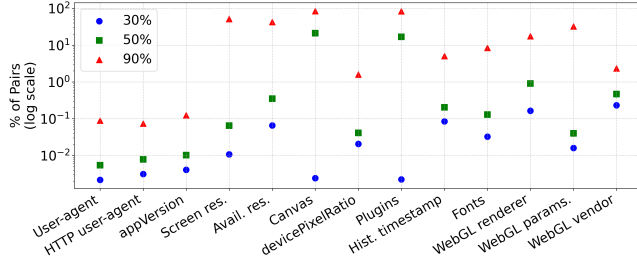


Figure 4: Percentage of unique pairs needed to cover 30%, 50%, and 90% of instabilities in Bank-A.

incorporate such features. Here, we further discuss such defenses, for completeness.

Anti-canvas fingerprinting. Privacy-focused browsers mitigate canvas fingerprinting by introducing noise. For instance, both Brave and Firefox (via its Enhanced Tracking Protection’s strict mode) apply minor RGB shifts to a small, random subset of canvas pixels, resulting in the canvas fingerprint hash being unique across all devices. To evaluate the impact in our analysis, we analyzed our largest dataset, Bank-A. Among the 2,537,810 LoginIDs that experienced a canvas fingerprint change between September and February, only 4.03% (102,221) exhibited a unique canvas fingerprint value. Even if *all* such cases are due to anti-fp defenses, which is unlikely, this defense only accounts for a small portion of the instabilities that we observe and analyze.

Anti-font fingerprinting. Brave browsers randomly exposes a subset of fonts to each site. However, for a given font list, Brave (v1.79.123) consistently returns the same subset across sessions (i.e., after browser restarts) when running in normal mode with default settings, with Shields enabled. We tested multiple sites using the same code and observed consistent results, which contradicts Brave’s claim of per-session font randomization. Although earlier versions exhibited per-session font randomization, our observations indicate that this behavior is no longer present. In contrast, the anti-font fingerprinting mechanisms in Tor and Firefox restrict the list of exposed fonts, leading to standardized results rather than unstable ones.

Anti-tracking tools causing user-agent instability. A factor contributing to user-agent changes beyond browser updates is the inclusion of extraneous product/version combinations. These changes are often caused by privacy tools, like AVG AntiTrack. These random product/version combinations typically appear in the format <Label>/XX.X.XXXX.X, appended to the end of a standard user agent string. For instance, Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36 Unique/93.7.7376.77. We observed varying labels like: “Unique,” “Agency,” “Config,” “OpenWave,” “GLS,” “AtContent,” “Viewer,” “Herring,” and “Trailer,” across all browsers. These labels can vary dynamically, appearing as “OpenWave” in one visit and “Config” in another, and may only appear in a subset of visits. In Bank-A’s dataset, we observed such labels in 7,807 devices (0.11%) over six months, indicating the limited adoption of such tools.

User-agent reduction. For the navigator.platform attribute, the most common change is from Linux armv8l to Linux armv8l

(notice the lowercase letter “l” changing to the numerical representation of the digit “one”). This change results from Chromium’s user-agent reduction [55], which standardizes the value to Linux armv8l on the Android platform to reduce identifying information. As such, nearly 99% of them change with a browser upgrade, and only 4% experience a value flip. This instability is primarily observed in WebViews; while the headers typically remain unchanged, the user-agent and platform values retrieved via JavaScript show inconsistency: some sessions reflect the reduced format, while others do not, leading to flip-flopping values. This is because unlike full browsers, WebViews embedded within apps adopt updates less uniformly. Among the devices exhibiting value flips, 20.2% are associated with MY_VZW_APP and include channelId: VZW-MFA-ANDROID in the user-agent, indicating that users were accessing the authentication workflow through the My Verizon app. Thus, in-app browsers may expose more information through the user-agent string and adopt privacy or security updates less consistently than full browsers.

Insight 4: Anti-fingerprinting. Only a small portion of attribute instabilities can be attributed to anti-fingerprinting defenses. Moreover, defense implementations may deviate from what browser vendors report.

3 ADAPTIVE FINGERPRINTING

Our extensive analysis of a large, real-world dataset of browser fingerprints revealed their inherently unstable nature. Our findings highlight new pitfalls for using browser fingerprints in practice, and have significant ramifications for the effectiveness of risk-based analysis. It is evident that simply combining the various fingerprinting attributes for generating an identifier (e.g., as done by FingerprintJS [21], the most popular fingerprinting library) will result in many instances of unsuccessful device re-identification due to unstable values (or software updates, as one would expect).

To mitigate the effect of instability, we propose a novel adaptive fingerprinting methodology that leverages our observations for re-identifying devices by “anticipating” unstable values that devices may exhibit in the future. This is done by building a knowledge base of associated pair-values that occur due to instability (i.e., *FP-flip* pairs). This strategy yields a fingerprinting system that adapts to attribute instability and is able to predict attribute changes and maintain accurate device re-identification capabilities despite the inherent instability of individual fingerprinting attributes.

In more detail, while fingerprinting attributes are unstable, they do not always generate random values. Instead, they can flip-flop between a set of values depending on the devices’ characteristics, and the value that may change in subsequent sessions can be predicted from the current value. For instance, in our Bank-A data from August, when the platform value was Linux armv8l, 10,859 user devices had a different value in subsequent sessions. Notably, 89.20% of these devices switched to either Linux armv8l or Linux aarch64, and the top seven different values accounted for 99.93% of the devices. This pattern was consistent across companies; in smaller companies the top value alone accounted for 100% of the changes. Similarly, for one specific fonts hash value, 23,511 devices experienced unstable values in subsequent sessions, with 99.83% of them changing to the same different value. As such, the motivation

behind our adaptive strategy is that by curating a set of possible FP-flip values based on a given device's initial value, we can predict most changes that occur in fingerprints. Exceptions to this prediction can occur in cases where major software upgrades introduce new attribute values not included in the current knowledge base.

User-driven changes. Some value pairs may contradict each other; e.g., 380 devices from Bank-A show an unrealistic platform change from Linux armv8l to Win32, indicating that the value was manipulated. We do not exclude such cases, as they reveal software-driven changes and should be included in the set of possible future values.

Generating the knowledge base. Our adaptive strategy utilizes a knowledge base of associated FP-flip values that are derived from a given dataset of benign users' browser fingerprints. While this knowledge base can be periodically re-generated (e.g., to account for major software updates), in our evaluation (§4) we use Bank-A's dataset collected in August to assess our adaptive strategy over the following six months and across all five companies. For each cookie, we track attribute changes across sessions and organize them into pairs. For example, if a WebGL vendor value changes from "Intel" to "Google Inc. (Intel)", this relationship is captured by combining these two values into an FP-flip pair. We then tally the occurrences of each pair across the dataset. If a pair is observed *in more than five user accounts*, it is added to our list of potential anticipated values for this initial value, for this specific attribute. We note that an initial value may change to multiple future values.

Optimized feature selection. Our datasets include 18 fingerprinting attributes, but we do not introduce anticipated values for every attribute. The most unstable attributes, HTTP user-agent, user-agent, and appVersion, frequently change with browser updates, introducing previously unseen values. Since these updates often alter the numerical components of these attributes, we retain only their alphabetic components. In addition, we use a UA parser [42] to extract more stable information, such as system family and browser family, from these user-agent strings to better characterize the changes. We compare attributes between headers and JavaScript, with progressively increasing detail. For example, we begin with the generic device type, then consider the browser family, and finally examine the full user-agent string. Version numbers for iOS and Android systems are excluded, as we observed that they tend to upgrade more frequently. Ultimately, we derived 28 features from the fingerprinting attributes and introduce anticipated values for these features.

Prediction workflow. Algorithm 1 shows the prediction workflow, where the VERIFYDEVICE function receives the browser fingerprints associated with an account during the login attempt. Using our knowledge base, we aim to re-identify the user device. The process begins with the initial account login attempt, during which we collect the browser fingerprints and extract our features (line 4). For each feature, we retrieve the anticipated values from the knowledge base specific to the current value, and combine them with the current value to form a value set (5-6). We then compute an identifier by using the value sets of all features (8). For each subsequent session by this user account, we extract the same features from the new fingerprinting values (11). If a feature's value falls within the predefined value set, we use that set for the feature (13-14). Finally, we compute a new identifier for this session based

on the updated values (17). If the new identifier matches the initial identifier, it confirms that the same device is being used; otherwise, it indicates a different device (18).

Algorithm 1 Identification Workflow

```

1: procedure VERIFYDEVICE( $\mathcal{A}, \mathcal{KB}$ )
2:   Initial Login:
3:   let  $Login_{init} \in \mathcal{A}$ 
4:    $FP_{init} \leftarrow extractFP(Login_{init})$ 
5:   for each Feature in  $FP_{init}$  do
6:      $valSet(Value_{init}) \leftarrow Value_{init} \cup \mathcal{KB}(Value_{init})$ 
7:   end for
8:    $identifier_{init} \leftarrow computeIdentifier(\{valSet\})$ 
9:   Subsequent Logins:
10:  let  $Login_{subseq} \in \mathcal{A}$ 
11:   $FP_{subseq} \leftarrow extractFP(Login_{subseq})$ 
12:  for each Feature in  $FP_{subseq}$  do
13:    if  $Value_{subseq} \in valSet(initValue)$  then
14:       $Value_{subseq} \leftarrow valSet(Value_{init})$ 
15:    end if
16:  end for
17:   $identifier_{subseq} \leftarrow computeIdentifier(\{Value_{subseq}\})$ 
18:  if  $identifier_{subseq} = identifier_{init}$  then Same device else Different device
19: end procedure

```

We note that while our evaluation uses cookies to associate sessions with devices as ground truth, this is not a fundamental requirement for Predictive-FP. Nonetheless, in practice, Predictive-FP can straightforwardly be integrated into any system that maintains session continuity, such as a risk-based authentication workflow triggered at login, where the initial fingerprint is recorded upon account creation and compared against subsequent logins. Moreover, our evaluation demonstrates that a knowledge base learned from one deployment (Bank-A) generalizes across all five companies (§4), suggesting that once constructed, it can be applied across deployments without requiring per-system persistent identifiers.

4 EVALUATING PREDICTIVE-FP

In this section we experimentally evaluate Predictive-FP, our adaptive fingerprinting strategy. We assess two aspects of Predictive-FP's capabilities: (i) whether we can accurately re-identify users using the same device over time, and (ii) distinguish between different devices.

Experimental setup. We use Bank-A's data from August to generate our knowledge base (see §3), which is then used by Predictive-FP on all five companies (see Table 1). In the case of Bank-A, due to the massive amount of data, we evaluate Predictive-FP's effectiveness in September, October, and February. For the remaining companies we evaluate Predictive-FP for every month within the 6-month period.

Baseline comparison. To accurately assess the effectiveness of our adaptive strategy, in every experiment we also compare with the current non-adaptive approach (referred to as *Baseline*), which essentially is comprised of the same set of browser fingerprinting attributes combined into a single identifier. This is the approach used by popular and prevalent fingerprinting systems in the wild. It is important to emphasize that our goal here is to evaluate our adaptive strategy in comparison to the existing non-adaptive strategy, and not to compare the baseline set of common fingerprinting attributes collected by these companies to other fingerprinting solutions. In practice, our adaptive strategy can be leveraged by any

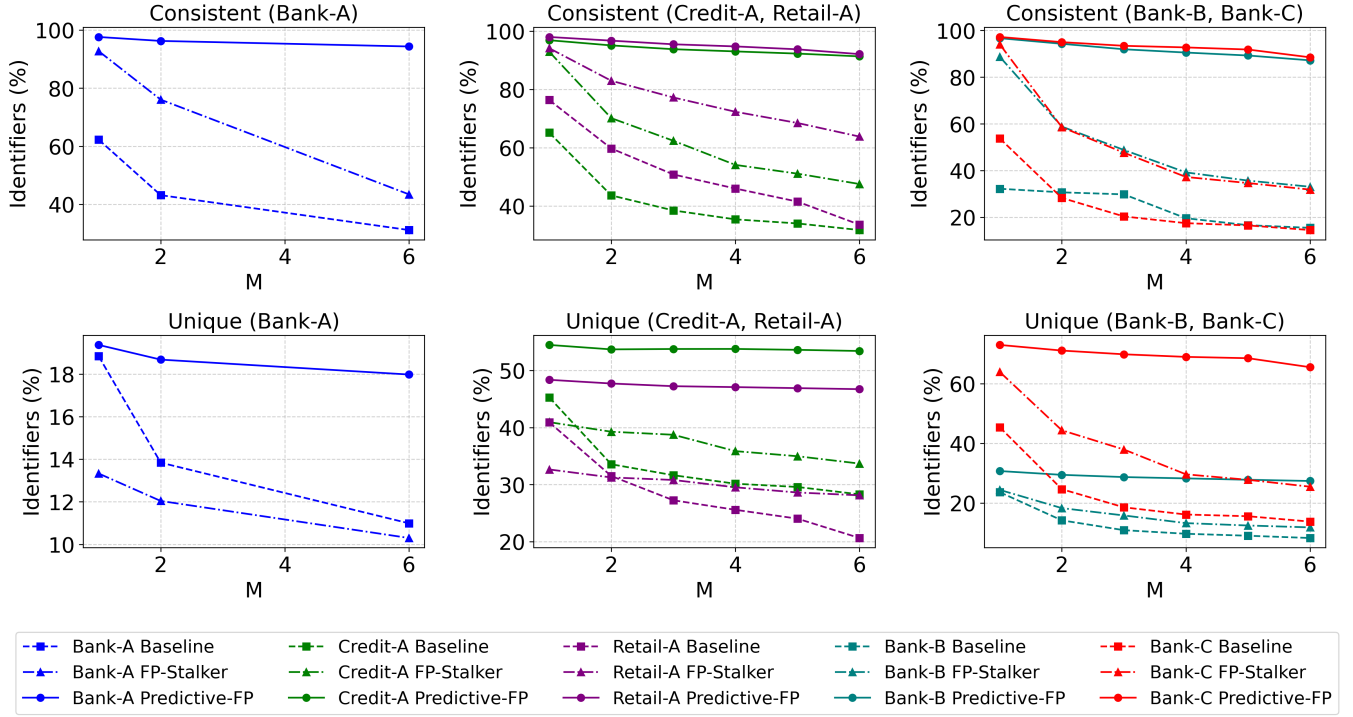


Figure 5: Device re-identification across five companies over six months.

fingerprinting system (e.g., FingerprintJS could employ our strategy using their own collection of attributes).

Evaluation metrics. To evaluate our system’s ability to re-identify user devices, we first identify the user by their LoginID. A user account may be associated with multiple cookies due to cookie clearing or the use of multiple devices. To ensure that only one device is evaluated per account, we select the cookie with the most sessions for a given account and remove the others. We define as a *Consistent ID* an identifier assigned by the system to a specific device across *multiple* sessions. This means that the system is able to generate the same identifier for the device each time, allowing for consistent re-identification. Therefore, a higher number of Consistent IDs within a given dataset signifies a system more effective at re-identifying devices. We also want to assess how effective the system is at differentiating between different devices, since different devices may have the same fingerprint-based identifier. As such, with *Unique IDs* we will denote the total distinct values of consistent identifiers generated by the system. The higher this number is for a given system, the better it is at distinguishing between devices.

Results. After building our knowledge base using the Bank-A August dataset, we run Predictive-FP and the Baseline on users’ recurring sessions across the remaining time period, for each separate month. Since we observe comparable results across all five companies, due to space constraints we only describe a subset of the experiments across the different companies. Overall, as can be seen in Figure 5, our results show that Predictive-FP significantly outperforms the Baseline across all companies, for both metrics, throughout the entire time period, demonstrating its higher discriminating power and long-term robustness in device re-identification.

Table 5: Bank-A device re-identification across six months. M denotes the month. Consistent# (%) and Unique# (%) denote the number and percentage of consistent and distinct consistent identifiers, respectively, among all returning users (6,838,475).

M	Baseline	FP-Stalker	Predictive-FP
<i>Consistent# (%)</i>			
1	4,259,686 (62.29%)	6,343,927 (92.77%)	6,677,087 (97.64%)
2	2,951,486 (43.16%)	5,201,371 (76.06%)	6,584,084 (96.28%)
6	2,130,185 (31.15%)	2,970,359 (43.44%)	6,453,649 (94.37%)
<i>Unique# (%)</i>			
1	1,288,369 (18.84%)	911,735 (13.33%)	1,324,613 (19.37%)
2	946,445 (13.84%)	823,237 (12.04%)	1,277,427 (18.68%)
6	751,548 (10.99%)	704,169 (10.30%)	1,229,837 (17.98%)

In Table 5 we can see that during the first month, Predictive-FP re-identifies 97.64% of users, compared to 62.29% with the Baseline approach, and uniquely identifies an additional 2.82% of devices compared to Baseline (1,324,613 versus 1,288,369). By the sixth month, Predictive-FP continues to outperform, re-identifying 94.37% of users, while Baseline sees a significant drop, re-identifying only 31.15%. Additionally, it uniquely identifies 63.65% more devices than the Baseline (1,229,837 versus 751,548).

Importantly, Predictive-FP’s performance is consistently effective across different companies. It re-identifies 96.92% of Credit-A’s users in the first month, compared to 64.55% with the Baseline approach, and uniquely identifies 54.48% of users versus Baseline’s

Table 6: Canvas image stability across Windows Chrome versions. ✓ indicates an unchanged image value. ✗ shows image value changes across versions.

Chrome Versions	Img 1	Img 2	Img 3	Img 4	Img 5	Img 6
120-125	✓	✓	✓	✓	✓	✓
125-126, 126-127	✓	✓	✗	✗	✓	✓
127-128	✓	✓	✓	✗	✓	✗
129-131	✓	✓	✓	✓	✓	✓
131-132	✗	✗	✗	✗	✓	✗

45.49%. By the sixth month, Predictive-FP still re-identifies 91.37% of devices and uniquely identifies 53.42%, whereas Baseline only re-identifies 31.49% of devices and uniquely identifies 28.23%, highlighting the enhanced performance of our adaptive strategy and the negative effect that instability has on existing approaches. We observed similar results for the remaining companies.

Canvas effect. Interestingly, we initially observed that Predictive-FP’s performance exhibited a sudden drop in the fourth month. This was due to the release of Chrome 120, which resulted in new canvas attribute values that were not part of our knowledge base. To quantify the overall impact of the canvas attribute, we repeat our measurement after removing it from the set of attributes. Surprisingly, we find that while removing the canvas attribute slightly reduces the unique identifiers during the first two months (it drops from 20.87% and 20.07%, to 19.37% and 18.68%), in the long-term our system produces *more* unique identifiers (17.98% compared to 14.96%). The positive effect of removing this attribute is far more pronounced when considering the number of consistent identifiers, especially after longer time periods. Indicatively, when using canvas, Predictive-FP produces consistent identifiers for 67.66% of users after six months, yet reaches 94.37% when the canvas attribute is dropped. For the non-adaptive baseline approach we observe a similar trend but with negligible value differences, i.e., using canvas does not significantly improve existing fingerprinting strategies over longer periods of time. Similar observations can be made for the remaining companies.

The fact that major software updates can introduce new canvas fingerprinting value reveals a limitation of implementations that rely on a single canvas image. To address this, we propose the use of multiple images. Our test site employs six canvas images, listed in Table 3, each generated with a distinct set of rendering techniques. As shown in Table 6, these images do not change their values simultaneously across browser versions. Such patterns can be efficiently learned and seamlessly integrated into Predictive-FP.

Instability reduction. Figure 6 calculates the average reduction per attribute across all companies. We compare the number of devices with unstable values for different features when using the traditional baseline approach and Predictive-FP, over the entire six month period. We note that the features labelled as browser letters, browser family, and system family are extracted from the user-agent strings. We observe a significant reduction in instability across most features when using our adaptive strategy; for instance, user-agent shows a reduction of 99.16%. Nonetheless, we do observe certain features that remain unstable even when

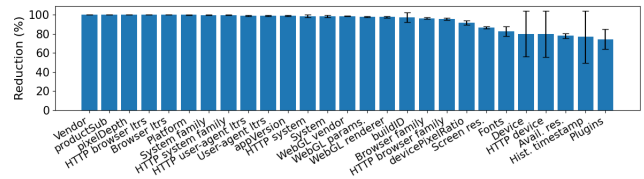


Figure 6: Predictive-FP’s reduction of unstable fingerprinting values over six months for the five companies.

Predictive-FP is used, such as the device-type features. Nonetheless, the two device-type features are more stable compared to others, with only 2,542 and 2,523 variations over six months at Bank-B. On the other hand, our system is able to handle 99.92% of the unstable platform values throughout the six-month period. This further corroborates our observations regarding the commonality of FP-flip value-pairs across different users, and the effectiveness of our knowledge-base generation methodology. Summarily, we see a significant reduction in instability across most features for the five companies. For most attributes our system achieves almost complete reduction, with a small number of attributes being less predictable and also exhibiting considerable deviation across different companies.

Identifier collisions. When used in real-world scenarios where web services have massive user bases, as expected, browser fingerprinting techniques can result in identifier collisions, i.e., different browsers running on different machines belonging to different users can produce the same device fingerprint. As shown in Figure 5, Predictive-FP consistently produces a higher percentage of unique identifiers than both the Baseline and FP-Stalker across all five companies throughout the six-month period. To get a better understanding of how identifier collisions are affected by Predictive-FP, we analyze the data from Bank-C in more detail, as a representative example of the effect of Predictive-FP. For each identifier generated by the system, we check whether it is shared by multiple LoginIDs. We find that 35.21% of LoginIDs are affected by identifier collisions in the Baseline approach, meaning their fingerprint-based identifier is shared with at least one other LoginID. Predictive-FP reduces this to 32.21%. The difference is more pronounced when examining the number of colliding identifiers: the Baseline produces 2,002 identifiers shared across Bank-C’s 4,340 distinct LoginIDs, whereas Predictive-FP produces only 672. At first glance, this result may seem counterintuitive, since Predictive-FP accepts a broader set of values per attribute, which could reduce discriminative power. However, this benefit can be traced back to the fingerprints’ inherent instability. In the Baseline case, instability fragments a single device’s fingerprint into many short-lived identifiers, each with reduced entropy. The proliferation of these transient identifiers increases the likelihood of accidental cross-device collisions. In contrast, Predictive-FP absorbs anticipated value changes into stable value sets, maintaining a single, high-entropy identifier per device rather than scattering it across fragments. Moreover, these value sets are attribute-specific and derived from population-level FP-flip patterns, which modestly expands the value-matching space while significantly reducing the fragmentation that drives collisions in the Baseline.

Comparison to prior work. Vastel et al. [57] explored how fingerprints evolve over time and proposed FP-Stalker, a rule-based strategy for linking fingerprints. To apply FP-Stalker to our dataset, we used the official implementation [49] and made the necessary modifications to ensure compatibility. Our guiding principle was to preserve the original design and logic without compromising its intended performance. Given that FP-Stalker was originally developed for a dataset collected nearly a decade ago, several adaptations were required to align it with today’s fingerprinting data. Specifically, we replaced deprecated Flash-based attributes with their JavaScript-based counterparts where available; otherwise, we skipped them. For example, the rule originally designed for Flash-based font fingerprinting was applied to JavaScript-based font fingerprinting in our implementation. Similarly, attributes present in our dataset but absent in the original FP-Stalker dataset (e.g., `availableResolution` and `pixelDepth`), were skipped during rule application. These skipped attributes were treated as always stable, which results in calculating a *better* linking performance for FP-Stalker than it would actually achieve in practice (see Appendix A.3 for the full list of adaptations).

As shown in Figure 5, FP-Stalker outperforms Baseline in re-identifying devices, achieving 92.77% accuracy in the first month but only 43.44% by the sixth month, highlighting the challenge posed by longer time periods. Moreover, while better than Baseline at re-identification, FP-Stalker has reduced discriminative power, as it uniquely identifies 5% fewer devices in the first month and 0.6% fewer in the sixth month. Importantly, for both metrics, Predictive-FP significantly outperforms FP-Stalker, particularly for longer time periods. Predictive-FP re-identifies 50.93% *more* devices and uniquely identifies 7.68% more devices than FP-Stalker. For the four smaller companies, as shown in Figure 5, FP-Stalker performs better than Baseline in both metrics. However, FP-Stalker’s advantage over Baseline diminishes with recurring visits. For instance, Retail-A’s devices have an average of 12.80 visits, FP-Stalker re-identifies 30.26% more devices than Baseline. Conversely, in datasets from the three companies Bank-B, Bank-C, and Credit-A, with over 40 visits per device, FP-Stalker’s improvement drops to $\approx 17\%$. In contrast, Predictive-FP is unaffected by the number of visits and consistently outperforms FP-Stalker across both metrics.

While Vastel et al. also proposed a hybrid approach that incorporates machine learning, it cannot scale to the size of our dataset. The original paper acknowledged the scalability issue of this approach, and subsequent work further reported that it could not scale to datasets with more than 300K fingerprints and characterized its performance as unacceptable for real-world deployment [30]. Moreover, the learning-based approach still applies the first three rules, which accounted for 56.39% and 61.08% of the failures in linking Bank-A’s data during the first and last month, respectively.

For our comparison, we do not use datasets from prior work as they are much smaller and outdated, and do not reflect the structure or behavior of current browsing environments. [57] contains only 15K fingerprints from less than 8K users, collected between 2015–2017, and includes many deprecated attributes that are no longer available or used today, and severely outdated values. Indicatively, it includes plugins like “nppdf32.dll” and “Facebook Video Calling Plugin”, that are no longer seen in modern browsers; moreover, Chrome and Firefox now return static, hardcoded plugin lists [35].

5 DISCUSSION AND MITIGATION

Implications of instability. We demonstrated how certain fingerprinting attributes inherently suffer from instability. The first-order effect of this is the inability to effectively and accurately re-identify user devices across sessions. In our analysis we found that the average stability window for attributes is typically in the order of 2–5 days. Depending on the actual application of the browser fingerprints, the ineffectiveness caused by instability can have additional implications. For instance, for anti-fraud or risk-based authentication system, this can lead to additional friction for users (e.g., additional security challenges like 2FA, transactions being blocked), and increased server-side computation (e.g., additional analysis of suspicious transactions).

Generalizability of findings. The fingerprinting attributes that we found to be unstable, are commonly used by a wide range of commercial and academic tools, and not unique to our dataset. Our controlled experiments have verified the use of problematic attributes even in mature and well-curated tools like FingerprintJS. While our dataset includes fewer attributes than AmIUnique [6], it captures the highest-entropy core used in practice [27]. Several of our attributes consolidate multiple entries from AmIUnique. For example, our two screen attributes correspond to AmIUnique’s four screen geometry attributes. Attributes absent from our dataset, such as Do Not Track and storage-related flags, offer comparatively lower entropy and are not collected by F5, Inc.’s production system. Thus, it is important to emphasize that our insights will similarly apply to the wider Internet public, and our adaptive approach can be readily used by any web service using common fingerprinting techniques. Crucially, instabilities are more likely to manifest for specific attribute and device combinations; as a result, observing these behaviors necessitates the analysis of extremely large, longitudinal real-world datasets (like ours). This, coupled with the natural evolution of fingerprints due to software updates, could explain why the inherently unstable nature of fingerprints has not been previously investigated.

Dataset properties. Our analysis is based on a large-scale dataset procured by F5, Inc. that offers security solutions to a wide range of services, including high-value financial institutions. The dataset is comprised of login requests that have been labelled as legitimate (i.e., it does not contain data from sessions flagged as bot-driven or fraudulent). Nonetheless, the dataset could still include a small number of such cases, e.g., from more sophisticated attackers that have managed to hijack users’ cookies *and* spoof their browser fingerprints [11, 32]. Moreover, as previously discussed, certain legitimate users may leverage defensive tools that randomize or spoof certain attributes. However, our knowledge-base construction strategy is designed to identify FP-flip attribute value-pairs that occur across multiple users. As such, our adaptive fingerprinting strategy is able to overcome the challenges posed by instability *and* existing anti-fingerprinting countermeasures that only change a limited number of attributes, and significantly outperform existing fingerprinting strategies (especially over longer time periods). Finally, while our system leverages common FP attributes, it could potentially be expanded to incorporate other attributes as well (e.g., list of installed browser extensions [29, 46]).

Implications for bot detection. Our findings characterize instability in legitimate traffic. An interesting future work direction is to explore whether bots exhibit different instability profiles, either overly stable due to scripted environments, or erratic due to fingerprint spoofing. Our characterization of natural FP-flip patterns could potentially serve as a baseline for distinguishing legitimate instability from bot-induced inconsistencies, complementing recent work that exploits cross-attribute fingerprint inconsistencies for detecting evasive bots [58].

Using stable signals. An alternative strategy for mitigating the effect of instability, is to focus on stable attributes. During our initial experiments, we focused on more stable signals, e.g., platform. The goal was to re-identify users using the minimal set of attributes, assuming that fewer attributes are less likely to introduce instability. We also leveraged uncommon attribute combinations to improve discriminative power, e.g., a macOS user browsing with Edge browser, while discarding signals that tend to vary over time. However, this strategy significantly reduced the fingerprint’s discriminative power, especially in services with large user bases. While it achieved high Consistent ID scores, the Unique ID metric dropped considerably due to the limited entropy. In contrast, Predictive-FP strategy maintains a high Consistent ID metric while also preserving the entropy necessary to achieve strong Unique ID performance.

Mitigating tracking. While our predictive strategy can significantly improve risk-based authentication in high-value services, it can also be misused in other sites for tracking users. As such, we developed a Chrome browser extension that strategically randomizes the main fingerprinting attributes on a per-session basis, in a way that preserves normal website functionality. Importantly, our extension defeats Predictive-FP by ensuring each session produces a fingerprint outside its anticipated value sets. The extension allows the user to configure it by providing a list for specific allowlisted domains for which fingerprints are not randomized, and also define the duration of “sessions” (e.g., 1 day). That way, users can benefit from enhanced authentication workflows on high-value services (e.g., banks), while mitigating privacy-invasive practices on all other websites.

The extension introduces minor and carefully controlled noise into each attribute so that the resulting fingerprints vary across sessions, while remaining compatible with websites’ expectations. When appropriate, the injected values follow patterns that resemble plausible system properties, such as realistic tokens inserted into the user-agent strings and WebGL parameters. For rendering-related attributes, including canvas pixels and screen geometry, the modifications are intentionally minimal to avoid visible artifacts or layout disruptions. Since a sophisticated tracker could attempt to detect our perturbations via cross-attribute consistency checks; we identify the main surfaces and explain how our design addresses them. For example, the HTTP header is aligned with `navigator.userAgent` and `navigator.appVersion`, and updates to `screen.width` are coordinated with corresponding updates to `screen.availWidth`. For screen-related attributes, the extension applies small and deterministic shifts of ± 5 pixels to width, height, `availWidth`, and `availHeight`, changes that remain visually imperceptible and do not interfere with the website’s layout decisions.

For user-agent, the extension generates a synthetic token composed of three to ten random alphabetic characters, with the first letter capitalized. A session-specific hash then determines where within the original string this token is inserted. For fonts, the extension perturbs layout measurements using a per-element stable delta, ensuring that repeated measurements of the same element within a session remain consistent, while still varying across sessions. WebGL tokens are constructed to preserve plausible vendor and platform combinations. This modification preserves the general structure and semantics expected by websites’ compatibility checks, while preventing cross-session linkage.

Evaluation. We evaluated our extension on 30 sites from the Tranco top 50 websites, and an additional financial service (Chase Bank). For each site, we created an account, logged in, navigated to various parts of the site, and tested the typical actions made available to users. All sites rendered correctly during navigation, and normal browsing functionality remained intact in all cases. Issues only occurred during account registration workflows for six sites; Facebook and Netflix flagged the browser as abnormal and required additional verification steps, such as solving CAPTCHA challenges. These issues disappear when we disable the User-agent random-string injection, while keeping all other fingerprinting attributes randomized. Google was the only site where we had to temporarily disable our extension during account creation, while all other website activities were successful without any issues. Importantly, all the services function normally for login operations as long as a valid cookie is present (e.g., when signing in to an existing account). Overall, the extension has minimal impact on website functionality; routine browsing and authenticated sessions operate as expected.

Unlike existing solutions, our extension handles a broader attribute set while preserving functionality: Brave omits WebGL spoofing, Firefox’s ETP leaves user-agent and screen resolution unchanged, and aggressive tools like Fingerprint Spoofer [13] cause functional breakage. Prior work such as PriVaricator [39] framed its core privacy goal as breaking cross-session linkability via controlled randomization, and evaluated the privacy-breakage tradeoff on popular sites. Our extension targets a broader set of modern fingerprint surfaces and supports domain allowlisting for facilitating risk-based authentication in trusted high-value services. We provide more details about the functionality and evaluation in Appendix A.4.

6 RELATED WORK

This section discusses related work on browser fingerprinting techniques and applications, and the instability and linkability of fingerprints across sessions.

Browser fingerprinting techniques. Eckersley [16] introduced the concept of browser fingerprinting and demonstrated its potential for uniquely identifying users. Subsequent studies [3, 20, 27, 28, 44] explored fingerprinting mechanisms and identified key attributes that contribute to uniqueness. Later work further expanded fingerprinting capabilities. Monaco [36] proposed using peripheral timestamps. Das et al. [14] explored the use of smartphone sensor data. Other studies [31, 54, 56] showed that the increasing complexity of CSS can be used for fingerprinting. Recent studies [19, 24, 25]

leveraged WebGPU rendering capabilities. Several studies investigated the fingerprintability of browser extensions [48] by examining their interactions with the DOM [47, 52], unnecessary code and resources [51], web-accessible resources [23], injected styles [29], and user-triggered actions [46].

Applications of fingerprinting. Prior work [5, 50, 60] has investigated how fingerprints can help prevent unauthorized account access. Andriamilanto et al. [7, 9] proposed attribute selection frameworks that balance security and usability for authentication. Wu et al. [59] focused on bot detection. Several studies [15, 26, 33, 43] have proposed using canvas fingerprinting to improve the security of authentication mechanisms. Meanwhile, other studies [32, 34, 45] have demonstrated user impersonation attacks that target risk-based authentication systems that use browser fingerprints. Comparatively, our paper provides a large-scale real-world view of fingerprinting use in authentication workflows, and proposes a technique that can enhance risk-based authentication. Fingerprinting is also used for user tracking; several studies [2, 12, 18, 40] have demonstrated how fingerprinting enables cookieless user identification across sessions and websites. Subsequent studies [4, 17, 22] have explored use for personalization and targeted advertising.

Browser fingerprint instability. Andriamilanto et al. [8] reported that 90% of 216 fingerprinting attributes remained unchanged over six months. The study does not clarify what this significantly larger set includes compared to the 17 used by the popular library [6], or which attributes remain stable. Prior work [9] also reported an average of 2.83 changes out of 253 attributes between consecutive observations. This work focused on attribute selection and did not specify the observation interval. Vastel et al. [57] found that 50% of browser fingerprints change within five days and 80% within ten days, revealing substantial instability. Similarly, Li and Cao [30] observed that only one-third of fingerprints remained stable across multiple visits. Collectively, these studies suggest that fingerprint instability is primarily caused by software and system updates, user behavior, and changes in the computing environment. To the best of our knowledge, our paper presents the first large-scale, real-world study of the inherent instability of browser fingerprints, even when a device’s software and configuration remain unchanged.

Browser fingerprint linkability. FP-Stalker [57] is the first study to examine the evolution of browser fingerprints and proposed rule-based and learning-based approaches to track users for 54.48 days, with 26% remaining linkable for over 100 days. After that, Pugliese et al. [41] proposed an approach using feature stemming and optimization. The method increases the mean stability of linkable fingerprints to 10.7–11.9 weeks on the FP-Stalker dataset. However, their evaluation relies on a small dataset of fingerprints from 1,304 users, which does not capture the broad variability of browser fingerprints. Additionally, some attribute values (e.g., WebGL renderer) can change significantly and are not addressed by feature stemming. Finally, attribute reduction causes entropy loss, making the approach less effective for real-world deployment, particularly in scenarios with large user bases. Laor et al. [24] proposed integrating a GPU-based fingerprinting technique into FP-Stalker’s machine learning model, which increased the median tracking duration from 17.5 to 28 days. Similarly, their evaluation used a small dataset of fingerprints from 2,550 devices and relied on FP-Stalker’s

model, which has limited scalability. Moreover, the method required specialized GPU-based fingerprinting, which may not be practical in all environments. As a comparison, our approach is evaluated at a much larger scale and introduces the first adaptive device re-identification strategy that significantly improves re-identification performance without the scalability limitations of existing machine learning methods.

7 CONCLUSION

Browser fingerprinting has become a prevalent practice across modern websites, both as part of high-value services’ authentication workflows and anti-bot detection, as well as for tracking purposes. In this paper, we presented the first exploration of an unexpected, inherent limitation of browser fingerprinting techniques; that of *instability*. Our in-depth analysis of the browser fingerprints from millions of real-world web requests, coupled with our own experiments in controlled settings, demonstrated the fickle nature of browser fingerprinting attributes while also revealing specific patterns within their instability. Interestingly, the complex nature of browser engines and their interactions with the underlying operating system can lead to variations in the reporting of even the most straightforward system characteristics (e.g., the screen resolution). Our observations pose severe implications for the robustness and applicability of existing browser fingerprinting systems in real-world settings. To that end, we proposed an adaptive device re-identification strategy that leverages the instability patterns observed during our analysis, thereby enabling robust and long-term device differentiation and re-identification. Accordingly, we have released our code to facilitate additional research in the space, and also allow web services to immediately enhance their authentication systems. Nonetheless, due to the potential privacy risks posed by our technique, we also release an attribute randomization extension as a protection for end users.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and shepherd for their valuable feedback. This work was supported by the National Science Foundation under grants CNS-2211574 and CNS-2143363. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the NSF. This research used resources of the Center for Institutional Research Computing at Washington State University.

REFERENCES

- [1] [n. d.]. Predictive-FP – Source Code. <https://github.com/xlin29/predictive-fp>.
- [2] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 674–689.
- [3] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1129–1140.
- [4] Nasser Mohammed Al-Fannah, Wanpeng Li, and Chris J Mitchell. 2018. Beyond cookie monster amnesia: Real world persistent online tracking. In *Information Security: 21st International Conference, ISC 2018, Guildford, UK, September 9–12, 2018, Proceedings 21*. Springer, 481–501.
- [5] Furkan Alaca and Paul C Van Oorschot. 2016. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of the 32nd annual conference on computer security applications*. 289–301.
- [6] AmlUnique. [n. d.]. AmlUnique. <https://amiunique.org/>. Accessed: 2025-11-20.

- [7] Nampoina Andriamilanto and Tristan Allard. 2021. Brfast: a tool to select browser fingerprinting attributes for web authentication according to a usability-security trade-off. In *Companion Proceedings of the Web Conference 2021*. 701–704.
- [8] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2021. “Guess Who?” Large-scale data-centric study of the adequacy of browser fingerprints for web authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 14th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2020)*. Springer, 161–172.
- [9] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2020. FPSelect: low-cost browser fingerprints for mitigating dictionary attacks against web authentication mechanisms. In *Proceedings of the 36th Annual Computer Security Applications Conference*. 627–642.
- [10] Michael Bailey, David Ditttrich, Erin Kenneally, and Doug Maughan. 2012. The menlo report. *IEEE Security & Privacy* 10, 2 (2012), 71–75.
- [11] Michele Campobasso and Luca Allodi. 2020. Impersonation-as-a-Service: Characterizing the Emerging Criminal Infrastructure for User Impersonation at Scale. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*.
- [12] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-) browser fingerprinting via OS and hardware level features. In *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society.
- [13] Chrome Web Store. [n.d.]. Fingerprint Spoofer Extension. <https://chromewebstore.google.com/detail/fingerprint-spoofers/facgnnelgcipeopfbjcaipahbhdjgpc>. Accessed: 2025-11-20.
- [14] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The web’s sixth sense: A study of scripts accessing smartphone sensors. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1515–1532.
- [15] Nor Izyani Daud, Galoh Rashidah Haron, and Siti Suriyati Syd Othman. 2017. Adaptive authentication: Implementing random canvas fingerprinting as user attributes factor. In *2017 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE, 152–156.
- [16] Peter Eckersley. 2010. How Unique is Your Web Browser?. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies (PETS’10)*.
- [17] Amr ElBanna and Nashwa Abdelbaki. 2018. Browsers fingerprinting motives, methods, and countermeasures. In *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 1–5.
- [18] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 1388–1401.
- [19] Ethan Ferguson, Adam Wilson, and Hoda Naghibijouybari. 2024. Webgpu-spy: Finding fingerprints in the sandbox through gpu cache attacks. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. 158–171.
- [20] David Fifield and Serge Egelman. 2015. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26–30, 2015, Revised Selected Papers 19*. Springer, 107–124.
- [21] FingerprintJS. [n.d.]. FingerprintJS. <https://github.com/fingerprintjs/fingerprintjs>. Accessed: 2025-11-20.
- [22] Alejandro Gomez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 world wide web conference*.
- [23] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting.. In *In Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*.
- [24] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clémentine Maurice, Yossi Oren, Romain Rouvov, Walter Rudametkin, and Yuval Yarom. 2022. Drawnapart: A device identification technique based on remote gpu fingerprinting. *arXiv preprint arXiv:2201.09956* (2022).
- [25] Tomer Laor and Yossi Oren. 2025. Poster: LockedApart: Faster GPU Fingerprinting Through the Compute API. In *Proceedings of the Microarchitecture Security Conference*.
- [26] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. 2019. Morelian analysis for browsers: Making web authentication stronger with canvas fingerprinting. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*. Springer, 43–66.
- [27] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)* 14, 2 (2020), 1–33.
- [28] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE.
- [29] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. 2021. Fingerprinting in style: Detecting browser extensions via injected style sheets. In *30th USENIX Security Symposium (USENIX Security 21)*. 2507–2524.
- [30] Song Li and Yinzhi Cao. 2020. Who touched my browser fingerprint? a large-scale measurement study and classification of fingerprint dynamics. In *Proceedings of the ACM Internet Measurement Conference*. 370–385.
- [31] Xu Lin, Frederico Araujo, Teryl Taylor, Jiyong Jang, and Jason Polakis. 2023. Fashion Faux Pas: Implicit Stylistic Fingerprints for Bypassing Browsers’ Anti-Fingerprinting Defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 987–1004.
- [32] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. 2022. Phish in sheep’s clothing: Exploring the authentication pitfalls of browser fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*. 1651–1668.
- [33] Zengrui Liu and Nitesh Saxena. 2024. Dual Study of Canvas Fingerprinting Based Authentication: A Novel Spoofing Attack and the Countermeasure. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1354–1364.
- [34] Zengrui Liu, Prakash Shrestha, and Nitesh Saxena. 2022. Gummy browsers: targeted browser spoofing against state-of-the-art fingerprinting techniques. In *International Conference on Applied Cryptography and Network Security*. Springer, 147–169.
- [35] MDN Web Docs. [n.d.]. Navigator: plugins property. <https://developer.mozilla.org/en-US/docs/Web/API/Navigator/plugins>. Accessed: 2025-11-20.
- [36] John V Monaco. 2022. Device fingerprinting with peripheral timestamps. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1018–1033.
- [37] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. 2011. Fingerprinting information in JavaScript implementations. *Proceedings of W2SP 2*, 11 (2011).
- [38] Mozilla. [n.d.]. StandardFonts-win10.inc in Mozilla Central. <https://searchfox.org/mozilla-central/source/gfx/thebes/StandardFonts-win10.inc>. Accessed: 2025-11-20.
- [39] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*. 820–830.
- [40] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*. IEEE.
- [41] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. 2020. Long-term observation on browser fingerprinting: Users’ trackability and perspective. *Proceedings on Privacy Enhancing Technologies* (2020).
- [42] PyPI. [n.d.]. Python user agents. <https://pypi.org/project/user-agents/>. Accessed: 2025-11-20.
- [43] Esteban Rivera, Lizzy Tengana, Jesús Solano, Christian López, Johana Flórez, and Martín Ochoa. 2022. Scalable and secure html5 canvas-based user authentication. In *International Conference on Applied Cryptography and Network Security*. Springer, 554–574.
- [44] Takamichi Saito, Koki Yasuda, Takayuki Ishikawa, Rio Hosoi, Kazushi Takahashi, Yongyan Chen, and Marcin Zalasiński. 2016. Estimating CPU features by browser fingerprinting. In *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. IEEE, 587–592.
- [45] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. 2024. The double edged sword: identifying authentication pages and their fingerprinting behavior. In *Proceedings of the ACM Web Conference 2024*. 1690–1701.
- [46] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. 2022. The dangers of human touch: fingerprinting browser extensions through user actions. In *31st USENIX Security Symposium (USENIX Security 22)*. 717–733.
- [47] Konstantinos Solomos, Panagiotis Ilia, Nick Nikiforakis, and Jason Polakis. 2022. Escaping the confines of time: Continuous browser extension fingerprinting through ephemeral modifications. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2675–2688.
- [48] Konstantinos Solomos, Nick Nikiforakis, and Jason Polakis. 2024. Harnessing multiplicity: Granular browser extension fingerprinting through user configurations. In *2024 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 161–174.
- [49] Spirals Team. [n.d.]. FPSalker. <https://github.com/Spirals-Team/FPSalker>. Accessed: 2025-11-20.
- [50] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2015. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proceedings of the eighth European workshop on system security*. 1–6.
- [51] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *The World Wide Web Conference*. 3244–3250.
- [52] Oleksii Starov and Nick Nikiforakis. 2017. Xhound: Quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE.
- [53] Stat Counter. [n.d.]. Browser Market Share. <https://gs.statcounter.com/browser-market-share>. Accessed: 2025-11-20.

- [54] Naoki Takei, Takamichi Saito, Ko Takasu, and Tomotaka Yamada. 2015. Web browser fingerprinting using only cascading style sheets. In *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. IEEE, 57–63.
- [55] The Chromium Projects. [n. d.]. User-Agent Reduction. <https://www.chromium.org/updates/ua-reduction/>.
- [56] Leon Trampert, Daniel Weber, Lukas Gerlach, Christian Rossow, and Michael Schwarz. 2025. Cascading Spy Sheets: Exploiting the Complexity of Modern CSS for Email and Browser Fingerprinting. (2025).
- [57] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE.
- [58] Hari Venugopalan, Shaoor Munir, Shuaib Ahmed, Tangbaihe Wang, Samuel T King, and Zubair Shafiq. 2025. Fp-inconsistent: Measurement and analysis of fingerprint inconsistencies in evasive bot traffic. In *Proceedings of the 2025 ACM Internet Measurement Conference*. 134–149.
- [59] Shujiang Wu, Pengfei Sun, Yao Zhao, and Yinzhi Cao. 2023. Him of Many Faces: Characterizing Billion-scale Adversarial and Benign Browser Fingerprints on Commercial Websites.. In *NDSS*.
- [60] Yiming Yan, Haiyong Zhao, and Haipeng Qu. 2024. A Browser Fingerprint Authentication Scheme Based on the Browser Cache Side-Channel Technology. *Electronics* (2079-9292) 13, 14 (2024).

A APPENDIX

A.1 Ethics

Datasets. The data used in our large-scale analysis (see §2) was procured by F5, Inc., which provides a security product for detecting and preventing bot activities and fraudulent transactions in the web services that we have included in our study. This data collection is already part of the companies’ security process, as it provides the data necessary for F5, Inc.’s security product to effectively prevent bots and fraudulent activities. The data provided to us only included the browser fingerprint attributes and anonymized meta-data (anonymized via secure one-way hashing by F5, Inc.) needed to differentiate the origin of the sessions. A detailed study protocol (including the nature and origin of the data to be collected and analyzed) was submitted to our IRB for review. The IRB reviewed the materials and determined that the study qualified for exemption, indicating that it posed minimal risk. We have also anonymized the names of the web services, and will delete the dataset upon publication, to avoid any unnecessary risk. The additional controlled validation experiments and root-cause analysis, were conducted on our personal devices.

Following the principles outlined in the Menlo Report (which adapted the principles of the Belmont Report), we conducted a stakeholder-based ethical analysis [10]:

Stakeholders. Stakeholders relevant to this research:

- (1) End users whose browsers may be fingerprinted.
- (2) Web services that rely on fingerprinting for authentication.
- (3) Developers of fingerprinting libraries and anti-tracking tools.
- (4) Trackers that aggregate fingerprint data for profiling or advertising.
- (5) Security and privacy research community.

Respect for persons. We note that our research does not directly involve human subjects and, as such, informed consent procedures are not applicable in our case. The analyzed dataset does not contain sensitive personal information, and our study focuses solely on associating various device-related attributes for differentiating and re-identifying devices. We do not derive any insights about or from human subjects’ behavior. Moreover, the data used in our analysis is already being collected by F5, Inc. as part of their customers’ anti-bot and anti-fraud security countermeasures, and is covered by appropriate agreements as well as the customers’ terms of service. Crucially, our study does not require any additional data collection than what already takes place in the respective services, nor is there any additional interaction with users that could cause undue harm or discomfort.

Beneficence. We have designed our research study so as minimize any potential harm to real websites and users, while maximizing the benefits that are provided by our findings. First, as aforementioned, our study was conducted on data that is already collected during the authentication process as part of the respective companies’ efforts to mitigate bots and fraudulent activities. F5, Inc. has a legal and ethics review department to ensure that all personal data is securely processed and stored, following established privacy laws like GDPR and CCPA. They also have agreements in place with their customers (i.e., web services included in our study) to ensure that their private data is securely stored and processed. Moreover, our observations from analyzing that dataset allowed us to develop

a more robust and effective fingerprinting strategy that will allow any web service to augment their authentication process. This will have direct benefits for end users and the web ecosystem at large.

Respect for Law and Public Interest. As aforementioned, all data collection conducted by F5, Inc. is vetted by their legal and ethics review department, and strictly adheres to established privacy laws. Moreover, specific agreements are in place with the webservices, governing the collection and storage of data.

Impacts. It is important to note that browser fingerprinting techniques are also used for tracking, advertising, and other privacy-invasive practices. Therefore, our predictive fingerprinting technique could potentially be misused; however, sophisticated or well-resourced tracking entities may already be aware of fingerprint instability and equipped to exploit it. To that end, we also develop and will open source a fingerprint-randomization extension that can be readily deployed by end users to protect themselves from such privacy-invasive practices, while allowing high-value services to continue employing fingerprinting to better secure user accounts and prevent fraud. As such, we believe that the publication of our research will have a net positive impact by providing a concrete mitigation tool for users. Importantly, our findings provide invaluable insight for additional research in anti-fingerprinting defenses, by both the research community and browser vendors.

Justice. The potential benefits of our work, which include enhancing authentication workflows in web services and providing users with the means to mitigate tracking, are broadly distributed, while the design of our study has minimized potential risks.

A.2 Attribute Instability Across Browser–Platform Combinations

Table 7 highlights devices (platform and browser) with unstable attributes across four companies. We find that the canvas attribute is extremely unstable in the Samsung Internet browser (the default browser in Samsung mobile devices) across all companies. Specifically, we observe instability in 88.5% of Linux aarch64 Samsung browsers at Bank-B (4,188 devices), 55.58% at Credit-A (2,118 devices), and 57.55% at Retail-A (282 devices). Additionally, other variants of Samsung browsers on different Linux architectures also suffer from high instability rates, with 53.99% of Linux armv7l and 49.46% of Linux armv8l at Retail-A.

The available resolution attribute is notably unstable on the MacIntel platform, as explained previously. For instance, 27.88% of MacIntel Chrome browsers at Bank-B (1,414 devices), and 20.29% at Retail-A (488 devices), exhibit multiple values over time. These variations typically have minimal pixel differences and are not due to users connecting an external monitor to a laptop. For instance, 7,054 devices at Bank-A and 30 at Bank-B experienced a change from 1440×807 to 1440×809 . The font attribute exhibits issues with Linux x86_64 Chrome and Win32 Firefox. At Bank-B, 28.10% of Linux x86_64 Chrome and 22.23% of Win32 Firefox instances exhibit instability for font fingerprinting.

The most unstable browser for pixelDepth across companies is MacIntel Chrome. For Bank-B, 5.28% of these browsers (268 devices) are unstable. More than 7% of Linux armv81 Chrome browsers exhibit unstable resolution values across all companies. Moreover, WebGL attributes exhibit varying degrees of instability.

Table 7: Top browsers exhibiting unstable attribute issues across Bank-A, Bank-B, Credit-A, and Retail-A. Browser refers to the combination of navigator.platform and browser family extracted from the user-agent. Unstable% (#) represents the percentage and number of cookies with unstable values. - indicates insufficient number of cookies with unstable values for a given browser (less than 1,000 for Bank-A or 20 for the other companies).

Attribute	Bank-A		Bank-B		Credit-A		Retail-A	
	Browser	Unstable% (#)	Browser	Unstable% (#)	Browser	Unstable% (#)	Browser	Unstable% (#)
Canvas	Linux aarch64 Samsung	85.15%(93,664)	Linux aarch64 Samsung	88.50%(4,188)	Linux armv81 Samsung	64.41%(38)	Linux aarch64 Samsung	55.58%(2,118)
	Linux armv81 Samsung	75.45%(13,265)	Linux armv71 Samsung	75.00%(36)	Linux aarch64 Samsung	57.55%(282)	Linux armv71 Samsung	53.99%(88)
	Linux armv71 Samsung	70.15%(1,147)	Linux armv81 Samsung	70.09%(157)	Linux armv81 Chrome Mobile	6.66%(315)	Linux armv81 Samsung	49.46%(231)
Avail. res.	Linux aarch64 Mobile Safari	28.14%(1,339)	MacIntel Chrome	27.88%(1,414)	MacIntel Safari	16.85%(1,466)	MacIntel Chrome	20.29%(488)
	MacIntel Chrome	28.05%(246,696)	MacIntel Firefox	22.24%(139)	MacIntel Firefox	14.51%(264)	MacIntel Safari	7.46%(849)
	MacIntel Safari	23.38%(286,337)	MacIntel Safari	19.63%(1,736)	MacIntel Chrome	7.45%(1,096)	Win32 Chrome	5.79%(465)
Screen res.	Linux aarch64 Mobile Safari	28.14%(1,339)	MacIntel Chrome	7.67%(389)	Linux armv81 Chrome	7.67%(27)	Linux armv81 Chrome	7.04%(35)
	Linux armv81 Mobile Safari	25.70%(1,131)	Linux armv81 Chrome	7.57%(57)	Win32 Chrome	5.99%(1,467)	Win32 Chrome	5.20%(418)
	Win32 Chrome	6.68%(250,679)	Win32 Chrome	7.51%(2,820)	MacIntel Firefox	3.02%(55)	MacIntel Chrome	4.32%(104)
devicePixel Ratio	Linux aarch64 Mobile Safari	25.82%(1,229)	MacIntel Chrome	5.91%(300)	Win32 Chrome	3.25%(796)	Win32 Chrome	3.12%(251)
	Linux armv81 Mobile Safari	24.39%(1,073)	Win32 Chrome	5.34%(2,004)	MacIntel Firefox	2.31%(42)	MacIntel Chrome	2.70%(65)
	Win32 Chrome	7.19%(270,220)	MacIntel Safari	4.12%(345)	Win32 Firefox	1.74%(110)	Win32 Edge	2.41%(73)
Fonts	Win32 Mobile Safari	25.61%(1,117)	Linux x86_64 Chrome	28.10%(428)	Win32 Firefox	13.37%(845)	Linux x86_64 Chrome	29.84%(131)
	Win32 Firefox	21.99%(112,530)	Win32 Firefox	26.40%(958)	Win32 Chrome	2.16%(530)	Win32 Firefox	1.16%(29)
	MacIntel Firefox	1.82%(2,239)	Win32 Chrome	24.68%(875)	Linux x86_64 Chrome	1.47%(245)	Win32 Chrome	0.66%(53)
WebGL renderer	Linux aarch64 Mobile Safari	27.55%(1,311)	Linux x86_64 Chrome	3.61%(55)	Linux x86_64 Chrome	6.83%(30)	Linux x86_64 Chrome	6.83%(30)
	Win32 Mobile Safari	26.64%(1,162)	MacIntel Chrome	1.24%(63)	Win32 Chrome	0.45%(170)	Win32 Chrome	0.62%(50)
	MacIntel Chrome	1.52%(13,390)	Win32 Chrome	0.45%(170)	Win32 Edge	0.41%(55)	Win32 Edge	0.41%(55)
WebGL vendor	Linux x86_64 Chrome	5.53%(9,111)	Linux x86_64 Chrome	1.64%(25)	Win32 Chrome	1.67%(410)	Linux x86_64 Chrome	5.01%(22)
	MacIntel Chrome	1.47%(12,942)	MacIntel Chrome	1.08%(55)	Linux x86_64 Chrome	0.41%(68)	Win32 Chrome	0.57%(46)
	-	-	Win32 Chrome	0.43%(164)	MacIntel Chrome	0.35%(51)	-	-
WebGL parameters	Linux x86_64 Chrome	5.52%(9,086)	Win32 Chrome	1.42%(348)	Win32 Chrome	1.42%(348)	Linux x86_64 Chrome	4.78%(21)
	Win32 Chrome	0.80%(29,961)	MacIntel Firefox	1.64%(30)	MacIntel Firefox	1.64%(30)	Win32 Chrome	0.56%(45)
	-	-	MacIntel Chrome	0.59%(30)	MacIntel Chrome	0.59%(30)	-	-
Plugins	iPhone Mobile Safari	3.33%(8,508)	MacIntel Chrome	2.11%(107)	Win32 Chrome	1.53%(374)	Win32 Chrome	2.35%(189)
	Win32 Chrome	2.16%(81,045)	Win32 Chrome	1.70%(637)	MacIntel Chrome	0.63%(92)	MacIntel Chrome	2.29%(55)
	Linux x86_64 Chrome	1.38%(2,277)	Win32 Firefox	1.62%(148)	Win32 Edge	0.33%(44)	Linux armv81 Chrome Mobile	0.93%(27)
Historical timestamp	iPhone Mobile Safari	5.09%(13,004)	iPhone Mobile Safari	5.47%(1,051)	Linux armv81 Chrome Mobile	2.60%(123)	Linux armv81 Chrome	2.60%(123)
	Linux armv81 Chrome Mobile	2.88%(23,206)	Linux aarch64 Samsung	4.37%(207)	iPhone Mobile Safari	2.40%(176)	iPhone Mobile Safari	2.40%(176)
	MacIntel Safari	2.46%(30,119)	Linux x86_64 Firefox	3.62%(165)	MacIntel Safari	1.51%(131)	MacIntel Safari	1.51%(131)

With Linux x86_64 Chrome browsers, 3.61% WebGL renderer and 1.64% WebGL vendor are unstable at Credit-A. Finally, we also find that while mobile browsers seem unstable for historical timestamp (which is the string representation of a historical timestamp in the browser used to capture the user’s timezone), this is typically due to user mobility.

A.3 FP-Stalker Attribute Adaptations

Applying FP-Stalker to our dataset required the following adaptations.

Substituted attributes. Flash-based attributes were replaced with their JavaScript-based counterparts where available. Specifically, fontsFlash was substituted with our JS-based fonts attribute, resolutionFlash with screen resolution, and platformFlash with platform. Flash-based languageFlash and languageInconsistency were skipped as no JS equivalent exists in our dataset.

Skipped: present in FP-Stalker but absent from our dataset. The following attributes exist in FP-Stalker’s ruleset but are not collected in our dataset and were therefore skipped: acceptHttp, languageHttp, orderHttp, connectionHttp, encodingHttp, dntJS, cookiesJS, sessionJS, IEDataJS, localJS, nbPlugins, and platformInconsistency.

Skipped: present in our dataset but absent from FP-Stalker. The following attributes are collected in our dataset but have no corresponding rule in FP-Stalker and were therefore not used: appVersion, buildID, devicePixelRatio, pixelDepth, available resolution, WebGL parameters, historical timestamp, productSub, and navigator.vendor.

All skipped attributes were treated as always stable, which yields a *more favorable* linking estimate for FP-Stalker than it would achieve in practice, since genuine changes in these attributes are simply ignored rather than penalized.

A.4 Mitigating Tracking Risk

Randomization strategy. In §5, we describe our privacy-preserving extension’s strategy towards randomizing specific browser fingerprinting attributes. Here, we provide details for additional attributes. For WebGL parameters, the extension intercepts calls to `getParameter()` on `WebGLRenderingContext` and `WebGL2RenderingContext` and appends a synthetic GPU-like token to vendor and renderer strings. For fonts, the extension perturbs layout measurements by occasionally introducing a one-pixel offset to `offsetWidth` and `offsetHeight`, which causes fingerprinting scripts to infer a different set of available fonts in each session. For canvas, the extension applies a value shift within ± 3 to the RGB channels of 5% of the pixels before calls to `toDataURL`, `toBlob`, and `getImageData`. These small, sparse changes leave the image visually intact while making any extracted bitmap or hash session-specific.

Evaluation. Table 8 summarizes the results of our experiments regarding whether any issues were observed when testing our extension with 30 of the Tranco top 50 websites. We supplemented our dataset with one additional site, as we wanted to include a financial service (Chase Bank) as well. For each site, we created an account, logged in, navigated to various parts of the site, and tested the typical actions that are available to users. All actions were performed with our defensive extension enabled. The only issues we detected throughout our experiments occurred during account

Domain	Functional
google.com	○
mail.ru	✓
microsoft.com	✓
facebook.com	○
cloudflare.com	✓
dzen.ru	✓
amazonaws.com	✓
instagram.com	○
youtube.com	○
apple.com	✓
office.com	✓
twitter.com	✓
live.com	✓
linkedin.com	✓
googlevideo.com	○
amazon.com	✓
workers.dev	✓
doubleclick.net	✓
azure.com	✓
wikipedia.org	✓
github.com	✓
bing.com	✓
digicert.com	✓
whatsapp.net	✓
fastly.net	✓
wordpress.org	✓
netflix.com	○
pinterest.com	✓
icloud.com	✓
googlesyndication.com	✓
chase.com	✓

Table 8: Website’s functional status when used with our defensive extension installed. ✓ indicates no breakage observed; ○ indicates the signup workflow being affected (e.g., CAPTCHA or security warning shown).

creation on certain sites. While these were easily resolvable in most cases (e.g., solving a CAPTCHA challenge or disabling only the user-agent randomization in our extension), Google showed a warning when creating a new Gmail account, that the browser “may not be secure”. In that case, we had to momentarily disable the entire extension to complete the account creation. However, all other activities were completed successfully with the extension enabled. In summary, our defensive extension allows users to browse the web unencumbered, allowing them to benefit from the additional account security offered by advanced risk-based authentication in high-value services (which they can trivially allowlist in the extension), while also preventing fingerprint-based tracking on all other websites.

While, other anti-fingerprinting solutions exist (e.g., built-in defenses in browsers, defensive browser extensions), none of them handle such an extensive set of features as ours, nor do they employ the strategic insertion of “noise” that our extension does. Indicatively, Brave does not spoof the user-agent string or the WebGL vendor and renderer values. It does, however, apply randomization to the plugins array. Our extension skips this attribute because modern browsers already return a fixed, non-informative plugin list. Firefox’s Enhanced Tracking Protection (ETP) strict mode likewise leaves important high-entropy surfaces untouched. In particular, it does not modify the user-agent string or screen resolution. The popular extension Fingerprint Spoofer [13] takes an aggressive approach to attribute spoofing, which causes functional breakage on

websites. For example, canvas images cannot be rendered correctly when the defense is enabled. Moreover, it introduces inconsistencies by spoofing the JS user-agent value while leaving the header user-agent unchanged, a mismatch that many websites can easily detect.