Large-Scale Measurement and Real-World Mitigation of Web Browser Fingerprinting in the Wild

Tom Ritter
Mozilla Corporation
San Francisco, USA
tritter@mozilla.com

Fatih Kilic
University of Toronto
Toronto, Canada
f.kilic@mail.utoronto.ca

Frederik Braun
Mozilla Corporation
San Francisco, USA
fbraun@mozilla.com

Elisa Luo
UC San Diego
San Diego, USA
e4luo@ucsd.edu

Geoffrey M. Voelker
UC San Diego
San Diego, USA
voelker@ucsd.edu

Stefan Savage
UC San Diego
San Diego, USA
ssavage@ucsd.edu

Christoph Kerschbaumer
Mozilla Corporation
San Francisco, USA
ckerschbaumer@mozilla.com

Abstract

Web browser fingerprinting has become a key tracking mechanism on the modern web, yet its real-world effectiveness and which of the existing mitigations offer value at scale remain insufficiently understood.

In this paper, we present a large-scale study of browser fingerprinting based on real-world telemetry from almost 25 million Firefox release users. This dataset enables us to quantify user uniqueness at population scale and identify which characteristics contribute most to this uniqueness.

Our analysis shows fingerprintability is largely driven by a small set of high-impact characteristics and their combinations, rather than the accumulation of many independent measurements. Building on these insights, we propose and evaluate fingerprinting protections based on their impact on uniqueness and cohort structure, and design data-driven mitigations.

These protections, shipped in Firefox 143, reduce the fraction of uniquely identifiable users from 75% to 46% while maintaining web compatibility, demonstrating that practical reductions in fingerprintability are achievable in a mainstream browser.

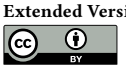
CCS Concepts

• **Security and privacy → Browser security**.

Keywords

Browser Fingerprinting, Large-Scale Measurement, Tracking Mitigation, Industry Deployment

Extended Version: October 6, 2026



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '26, November 15–19, 2026, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846585

ACM Reference Format:
Tom Ritter, Fatih Kilic, Frederik Braun, Elisa Luo, Geoffrey M. Voelker, Stefan Savage, and Christoph Kerschbaumer. 2026. Large-Scale Measurement and Real-World Mitigation of Web Browser Fingerprinting in the Wild. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 27 pages. https://doi.org/10.1145/3830454.3846585

1 Motivation

Today, all major web browsers deploy some mitigations to protect users against intrusive tracking.

In the early web, private browsing modes that cleared cookies and other client-side storage at the end of a session were largely sufficient to mitigate tracking [15, 48, 70, 92]. As cookie-based tracking became widespread, browser vendors introduced increasingly robust countermeasures, such as *Total Cookie Protection* in Firefox [82] and *Intelligent Tracking Prevention* in Safari [14].

Although these countermeasures are effective against traditional stateful tracking, they have also reshaped the tracking landscape, driving adversaries toward more sophisticated fingerprinting techniques based on stateless browser characteristics. Consequently, cookie-focused protections provide only limited resistance to pervasive cross-site tracking in practice.

In contrast to traditional stateful cookie-based tracking [1, 6, 19, 26, 29, 31, 32, 34, 35, 38, 55, 68, 69, 105, 128], fingerprinting is a stateless technique that exploits web APIs to collect browser settings and hardware characteristics, allowing adversaries to track users across sites.

In essence, browser fingerprinting leverages characteristics such as installed fonts, preferred languages, timezone, CPU information, and other measurements to construct an identifier, often unique, for a user's browser. As a result, users remain vulnerable to tracking with modern fingerprinting techniques approaching the reliability of traditional cookie-based tracking [65].

In response, browsers have begun deploying fingerprinting defenses, ranging from blocklists of known trackers to API-level noise injection or randomization (e.g., Web Audio [12] and Canvas [24]),

as well as standardizing responses to hide underlying system differences.

However, these approaches remain piecemeal: blocklists and targeted disruptions are susceptible to bypasses and require continuous maintenance, while API-level mitigations and standardization efforts are often developed in isolation for individual fingerprinting vectors. They lack a systematic understanding of which characteristics actually drive fingerprintability at scale. Without large-scale measurement to guide these decisions, browser vendors cannot reliably assess whether deployed defenses meaningfully reduce user uniqueness or merely address low-impact signals.

This lack of systematic understanding points to a fundamental gap: we do not yet have a clear, data-driven picture of which browser characteristics contribute most to fingerprintability in real-world populations, nor how existing defenses affect user uniqueness at scale.

Moreover, browser vendors face a fundamental trade-off: modifying web APIs to reduce fingerprintability often comes at the cost of user experience and web compatibility [54]. As an example, hiding useful system information from a web page might prevent it from functioning correctly (e.g., providing application downloads for the wrong operating system). But guarding a powerful feature behind a user prompt can lead to prompt fatigue, eventually causing only disruption and confusion rather than a privacy improvement [120].

We therefore analyze fingerprint data from real-world users, simulate commercial fingerprinting services, and develop data-driven protections.

Unlike prior large-scale studies of web user fingerprints, which rely on automated crawlers, volunteer samples, or users visiting a limited set of websites [8, 33, 45, 60], our approach captures real-world data at scale.

To the best of our knowledge, we present the first large-scale analysis of web fingerprinting using real-world telemetry from nearly 25 million Firefox users. While this dataset avoids common sources of bias in prior work (e.g., crawlers or self-selected volunteers), it is still subject to ecosystem-specific biases inherent to Firefox's user population. This dataset enables us to characterize the real fingerprinting surface on the web and provide detailed insights into which APIs are most valuable to fingerprinting services.

In detail, we contribute the following:

1) Large-Scale Measurement: We measure real-world, large-scale data from nearly 25 million Firefox release users about end users' hardware and software settings, covering characteristics that are exposed to the web, thereby making users fingerprintable. We analyze the gathered dataset to measure the entropy of characteristics and how they interact when combined into a fingerprint. Based on the reverse engineering of popular fingerprinting libraries, we calculate the number of users grouped into different-sized cohorts of shared fingerprints and demonstrate how many users are unique, as well as *why* those users are unique (Section 3).

2) Real-World Mitigation: Leveraging the insights of (1), we design and implement effective fingerprinting protection techniques which have since been deployed in Firefox (version 143). Evaluated against simulated commercial fingerprinting services, our techniques reduce user uniqueness from 75% to 46% while preserving web compatibility (Section 4).

3) Design Considerations: We present design considerations for deploying fingerprinting defenses, including entropy reduction, standardization, and randomization. We discuss how these techniques complement each other and their role in practical, deployable defenses (Section 5).

These results demonstrate that fingerprinting is not an unavoidable property of the web, but a tractable problem addressable through data-driven, deployable defenses.

2 Background

In this section, we establish fundamentals of web tracking, outline our threat model, discuss limitations of our approach, and describe the three JavaScript services we simulate to evaluate our protection techniques.

2.1 Tracking Fundamentals

All web browsers adhere to the guidelines of the same-origin policy [136] which, among other goals, protects users against the possibility of extensive tracking on the web. This fundamental browser security feature limits a website's access to information and hence establishes a first line of defense against tracking.

Cookie-Based Tracking: Cookie-based tracking enables websites to persist state in the otherwise stateless HTTP protocol [53]. While the data is often referred to as the Cookie Jar [128], it extends beyond HTTP cookies [18] to a broad range of explicit and implicit stateful identifiers, including mechanisms such as local storage and service workers [52, 121, 131, 132, 137].

While these mechanisms serve legitimate purposes, trackers exploit them to identify users across sites [1, 6, 19, 26, 29, 31, 32, 34, 35, 38, 55, 68, 69, 105, 128].

In response, browsers introduced mitigation mechanisms [14, 82, 112] that limit trackers' access to stateful identifiers in third party contexts and delete that state in different circumstances. As these defenses became widespread, tracking shifted toward alternative vectors, most notably browser fingerprinting.

Fingerprinting-Based Tracking: Browser fingerprinting is the capability of a site to observe software and hardware characteristics of an end user's browser and re-identify it across contexts, e.g., over time or across origins. It leverages measurements such as installed fonts, preferred languages, the results of mathematical operations, timezones, CPU information, and other characteristics to construct an identifier of a user's browser. While stateful tracking has at least some level of transparency to end-users (e.g., cookies can be viewed and deleted), fingerprinting is largely invisible to users, making it a more insidious tracking vector. While fingerprinting can serve legitimate purposes such as fraud detection and bot mitigation, the fundamental problem remains the same: it enables invasive tracking, making users prone to cross-site tracking attacks [8, 16, 44, 45, 54, 59, 60, 62, 64, 65, 108, 113, 119, 126, 127, 138].

2.2 Balancing Privacy and Usability

Most characteristics revealed about a user arise not from discretionary implementation choices made by the browser, but from web standards that require exposing this information or from unavoidable side effects of implementing those standards—such as differences in graphics rendering hardware. Trying to reduce the information revealed about a user typically involves deviations from standardized web APIs, which can introduce issues with web application functionality, referred to as *web compatibility*. To avoid this, two techniques are frequently proposed that shift the burden of web compatibility to the user.

Optional Stricter Controls: The most comprehensive, yet also most disruptive, fingerprinting defenses are implemented in Tor Browser [122] and the closely related Mullvad Browser [104]. Many of these protections have been upstreamed into Firefox, where they remain disabled by default [98]. Firefox's *Resist Fingerprinting (RFP)* mode [93] reproduces most of Tor Browser's defenses, but requires users to enable hidden configuration flags that place the browser into an unsupported mode.

These strict protections introduce substantial visible and functional regressions (i.e., web compatibility issues). Mozilla has received numerous reports documenting degraded browsing experiences [73], alongside a steady stream of user confusion in support channels where breakage is often not attributed to the enabled protections [2–5, 10, 71, 101–103, 124, 125]. In the period before our first Phase 1 protection (cf. Table 2) shipped in Firefox 115, 424 breakage reports were cataloged against Firefox itself [81].

As examples of strict protections with severe impact: in RFP mode all users' timezones are standardized to UTC (breaking calendaring apps); access to rendered graphics output is disabled, breaking functionality such as image cropping; and subtle clock precision changes cause failures in applications such as HTML5-based games.

Users are more likely to disable such protections than tolerate disruptions [91], and support data shows that even users who explicitly enable this mode often do not associate resulting breakage with the protections. While this level of protection may be acceptable for a niche browser like Tor, it is not viable for a mainstream browser such as Firefox, *even when presented as an option users must choose to enable.*

We note, however, that the bug reports Mozilla receives about this mode form a valuable catalog of real-world web compatibility constraints, helping inform the design of usable, practical, and deployable defenses that meet the requirements of a mainstream browser.

Prompting the User: Browser vendors generally avoid prompting users for security decisions, as it often reduces rather than improves security [20]. Prompts rely on users making informed, consistent choices, but in situations like fingerprinting, which are complex and ambiguous, users are unlikely to make knowledgeable decisions. Users also experience prompt fatigue: frequent requests lead to habitual approval, reducing prompts to noise rather than security checkpoints.

2.3 Threat Model

We assume the adversary can (1) execute JavaScript within web pages visited by the user, and (2) invoke browser-exposed APIs to collect information about the user's environment.

We do not assume that the adversary can exploit browser vulnerabilities or bypass existing security boundaries (e.g., the same-origin policy or iframe sandbox restrictions).

Instead, **our focus is on limiting the abuse of legitimate JavaScript APIs for fingerprinting purposes.**

2.4 Scope and Limitations

For our study (Section 3) and proposed protections (Section 4), we outline the scope of our approach and its limitations. These limitations fall outside our threat model or, in the case of web compatibility, limit our available actions, but are important for interpreting our results.

Web Compatibility Constraints: Fingerprinting defenses must preserve the user experience and avoid web compatibility issues. Prior work shows that more than half of users do not report breakage they encounter [109], so reported incidents represent only a fraction of actual breakage. Firefox's deployment process therefore places strict constraints on candidate mitigations. Before code ships, *design review* assesses each proposal against *historical web compatibility reports*—the breakage corpus filed against RFP—and against domain expertise on real-world API usage. Surviving proposals are deployed as *staged experiments* through Mozilla's Experimenter platform [80], which randomly enrolls a fraction of the release population and evaluates pre-declared metrics for statistical significance. We monitor *usage metrics* for abandonment or decline and *user feedback* submitted through Bugzilla and Firefox's feedback form. For mitigations that introduce observable differences, acceptability depends on impact severity, site popularity, available workarounds, and the likelihood of ecosystem adaptation. **Web compatibility is thus a hard constraint: strategies likely to cause unacceptable breakage are eliminated early** in the development lifecycle.

IP-Based Tracking: Fingerprinters can execute client-side JavaScript and send data to tracker-controlled servers [67], revealing the user's IP address as an additional tracking vector. Mitigating IP-based tracking falls outside the scope of our threat model and is not addressed by the protections presented in this paper.

Website Population: Our results reflect the global user population, not the population of any individual web site. While we can identify users as unique or part of a cohort in our dataset, this may not hold for a specific site's audience. For example, a Mandarin-localized user may be common globally, but unique on a local Icelandic website.

Bots: Autonomous programs (bots) may interact with websites and submit excessive telemetry data, leading to overcounting of certain characteristics. We assume that default telemetry settings in automated tools, together with safeguards in data collection pipelines (e.g., rate limiting and filtering of anomalous or duplicate submissions), sufficiently limit this effect.

Excluding Users from Data Collection: Our study relies on Firefox telemetry [96] (cf. Section 3.1), which collects data from

real-world users with minimal selection bias and includes an opt-out mechanism; we estimate that about 7% of users disable it. While this opt-out may introduce some bias, we expect its impact to be limited, as the dataset is neither self-selecting nor strongly regionally skewed.

Web Extensions: Fingerprinting libraries can detect content blockers [40, 42], introducing entropy. Roughly 40% of users install extensions [83], making them a common aspect of real-world browser configurations [58, 118]. However, incorporating extensions into our analysis poses practical and ethical challenges: accurate characterization would require collecting fine-grained, highly dynamic data (e.g., installed extensions, versions, and configurations), which is difficult to measure at scale. As extensions are not uniformly available across browsers and are largely outside browser vendors' control, we treat them as out of scope and part of the ambient fingerprinting surface.

2.5 Simulated Fingerprinting Services

For our evaluation, we reverse engineer and simulate three leading commercial services that allow us to assess how fingerprintable end users are. We selected these services based on their prevalence in the most recent measurement study [66] and their coverage in prior research [36, 106, 110, 116].

Fingerprint Pro: *Fingerprint Pro* [37] is a server-side, commercial browser fingerprinting service built upon the widely-used open-source *FingerprintJS*. *Fingerprint Pro* uses more sophisticated tracking techniques than its source-available counterpart, including additional fingerprinting surfaces such as MathML and emoji [67].

Akamai: *Akamai* has a broad reach over websites [66] and performs browser fingerprinting as part of its Bot Manager product [7]. While *Akamai's* Bot Manager uses fingerprinting techniques such as canvas fingerprinting to reduce web scraping and other malicious Web automation, we found that its privacy statement does not expressly disallow the repurposing of user browser data collected through its Bot Manager service for other purposes (i.e., tracking).

Signifyd: *Signifyd* [117] is a prominent [66, 116] fraud prevention service for e-commerce that distinguishes legitimate shoppers from malicious actors. Notably, analysis of its script suggests it is largely derived from *CreepJS* [30], another popular source-available fingerprinting library.

Fingerprinter Simulation: For each commercial service, we reconstruct the client-side measurements collected by its fingerprinting script using static and dynamic analysis. Because the server-side matching logic is proprietary, we do not replicate the exact identification algorithm. Instead, we model the *information available* to each service by combining all measurements into a canonical fingerprint.

Specifically, we compute a deterministic hash over these measurements, representing an *idealized matching model* that uses all signals consistently and without noise. This abstraction provides an upper bound on fingerprinter distinguishability and enables consistent comparisons across services and protection techniques.

This captures each service's *capability* to track users rather than its behavior—a distinction that matters because how a company uses collected data internally cannot be externally audited. Although fingerprinting is often justified by fraud and abuse prevention, allowing it to persist unhindered enables user tracking and privacy violations. Purpose-built alternatives for fraud and abuse prevention exist or are under development, including passkeys and WebAuthn [129] for authentication and Private Access Tokens [11] and PACT [57] for abuse prevention.

We exclude the browser's outer and inner dimensions because benign actions such as window resizing make them unsuitable as persistent identifiers. We confirm this empirically by probing *Fingerprint Pro* following Luo et al. [67]: perturbing these values does not affect the returned visitor ID.

Using three established, widely deployed fingerprinting services enables a real-world evaluation of our analysis and provides empirical insights into the effectiveness of our proposed protections.

3 Large-Scale Measurement

In this section, we present insights from a large-scale analysis of web fingerprinting based on **data collected from nearly 25 million Firefox release users** on desktop and mobile (Windows 12.1m, macOS 2.0m, Android 8.9m, and Linux 1.8m). This measurement study collects telemetry that enables analysis from a highly privileged vantage point with visibility into underlying hardware characteristics. In Sections 3.1 and 3.2 we outline the method and reasoning behind our data collection procedure. In Sections 3.3 and 3.4, we find that the *method* by which user uniqueness is computed can significantly affect the results of such measurement studies, and report the uniqueness of user fingerprints at a large population scale. Finally, in Sections 3.5 and 3.6 we find that no individual measurement contributes highly to user uniqueness, but rather, the identifying power is concentrated in a small subset of measurements.

3.1 Background on Data-Gathering

The initial data collection phase was conducted in June 2025. Based on this data, we developed fingerprinting protection techniques (cf. Section 4, Phase 2) which we began deploying in Firefox 143. To reflect the current ecosystem, re-evaluate the impact of these mitigations, and assist us in developing additional mitigations, we performed a second, updated data collection for this study. Our results are based on this telemetry [96] gathered during a Firefox release cycle, from **March 25th, 2026 to April 16th, 2026**.[1]

As per Mozilla's policy, all collected **data adheres to *Mozilla's Data Privacy Principles* [88], which enforce strict privacy and ethical standards** (cf. Appendix A).

3.2 Collected Characteristics and Measurements

With few exceptions (Section 2.4), we collected a comprehensive set of features known or potentially useful for browser fingerprinting.

[1] Because data collection spanned Daylight Saving Time (DST) transitions in several regions, results were normalized to a consistent DST offset to prevent otherwise identical clients from differing due to submission date.

We were sure to include features that were used in source-available fingerprinting libraries [30, 37, 39], and prior academic work [8, 16, 27, 44, 45, 60, 61, 64, 65, 108, 110, 113, 119, 138]. Of course, the data collected included all of the browser APIs used by the three fingerprinting scripts we have reverse engineered for our evaluation: *Akamai*, *Fingerprint Pro*, and *Signifyd* (Section 2.5). More details on how we identified the browser features collected by each service are provided in Appendix F.

In total, we collected **133 user characteristics comprising 708 distinct measurements** per user. We define a **characteristic** as a *conceptual signal* (e.g., default system fonts) and a **measurement** as a *concrete observation* (e.g., default serif or `monospace` font).

The full list of characteristics and associated measurements is provided in Appendix G. Importantly, our dataset covers a **superset of characteristics *actively used* by real-world fingerprinting deployments.**

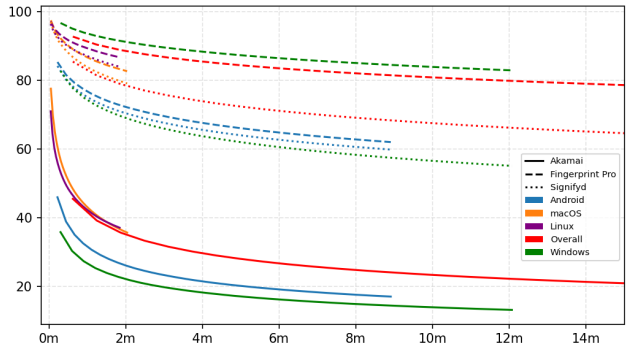


Figure 1: Effect of dataset size on fingerprint uniqueness. Y-axis shows percent of users classified as unique. Color indicates operating system, stroke style represents the evaluated fingerprinters.

3.3 Impact of Dataset Size and OS Segmentation

We first assess the impact of **dataset size** and **Operating System (OS) Segmentation** on user fingerprint uniqueness, demonstrating the importance of population size and composition.

Dataset size: For each of the three commercial fingerprinting services, we compute a per-user fingerprint from the measurements it collects. Then, we compute the number of users classified as unique by each commercial fingerprinting service over increasingly large *random subsets* of our dataset.

As shown on the x-axis in Figure 1, which represents the total number of users, the likelihood of a user appearing unique declines steeply as population size increases, across all fingerprinting services and operating systems. For example, uniqueness estimates fall sharply between 0 and 3 million users, with curves stabilizing only at larger scales, typically around 4 million users or more.

OS Segmentation: One of the characteristics revealed by the browser is the user's operating system, which browsers must report accurately for web compatibility. Misreporting the OS can cause significant issues, e.g., downloads for the wrong platform and broken website features such as non-functional shortcut keys [78]. The OS thus serves as a natural top-level differentiator for analyzing user populations. Partitioning users accordingly reveals substantial differences in both *how unique users are* and *why*.

As shown by lines with matching styles in Figure 1, the uniqueness percentages for each commercial fingerprinter can vary drastically for users of different operating systems. Specifically, for every fingerprinter there is at least a 23 percentage point difference across OSes. Only in three of the twelve scenarios does an individual OS's uniqueness closely match the overall result: macOS and Linux for *Akamai*, and macOS for *Signifyd*.

Key Insights: Our analysis shows that first, **dataset size fundamentally affects uniqueness estimates**: convergence for high-dimensional measurements requires datasets on the order of *millions* of samples, far exceeding prior work (cf. Section 6). Only at this scale do estimates stabilize, with additional data yielding negligible reductions in the proportion of unique users. Second, **population segmentation reveals substantial differences in uniqueness across user groups**. Thus, **aggregating heterogeneous populations without accounting for these differences can distort uniqueness estimates and lead to systematically misleading conclusions.** These findings have direct implications for future measurement studies: results obtained from a single platform or homogeneous population may not generalize, and studies should account for population composition when interpreting and reporting uniqueness.

3.4 Fingerprinting Uniqueness and Cohorts

Now, we evaluate the uniqueness and cohort structure of the user fingerprints in our dataset through the lens of real-world fingerprinting services. Similar to the previous section, we model the behavior of *Akamai*, *Fingerprint Pro*, and *Signifyd* by combining the measurements that each fingerprinter collects, and then grouping users sharing the same fingerprint into a *cohort*.

To the best of our knowledge, prior work [8, 51, 60] only reports the percentage of users that can be uniquely identified through their browser fingerprint (i.e., the percentage of users that are in a cohort of size 1). However, this does not tell the full story – two datasets may result in similar numbers of unique users while exhibiting substantially different cohort structures, leading to distinct privacy implications. As an extreme example, in one case all the non-unique users could share the same fingerprint, while in another the remaining non-unique users are all in cohorts of 2, meaning they share a fingerprint with only 1 other user.

Thus, we provide additional insight into the cohort structure of our dataset by reporting and comparing the overall cohort structure (i.e., the distribution of non-unique users across cohorts of varying sizes). To do this, we use the *Simpson Index*, a diversity metric that captures the concentration of users across cohorts and provides a single summary measure of the distribution. Intuitively, the ***Simpson Index* measures the probability that two randomly selected users share the same fingerprint**: a higher value indicates that users are concentrated into fewer, larger cohorts. As before, we partition all analysis by OS.

Service	Windows (12.1m) % Uniq	Windows (12.1m) Sim ($\times 10^{-7}$)	macOS (2.0m) %	macOS (2.0m) Sim	Android (8.9m) %	Android (8.9m) Sim	Linux (1.8m) %	Linux (1.8m) Sim	Weighted Average %
Fingerprint Pro	82.9%	3.16	82.6%	11.94	62.0%	46.24	86.7%	76.05	75.7%
Signifyd	55.1%	31.80	79.1%	13.40	59.8%	51.12	83.9%	80.70	60.9%
Akamai	13.1%	1653.90	35.7%	135.48	17.0%	581.77	37.1%	478.27	18.2%

Table 1: Baseline user uniqueness and cohort score across three real-world fingerprinting services, segmented by OS.

Baseline Uniqueness Across Fingerprinters: Table 1 summarizes the baseline uniqueness results for the three fingerprinting services, before any privacy features are applied. We observe substantial variation across services and operating systems, with *Fingerprint Pro* consistently yielding the highest proportion of unique users.

Specifically, it uniquely identifies a population-weighted average of 75.7% of users, with rates of 82.9% on Windows, 82.6% on macOS, 62.0% on Android, and 86.7% on Linux, indicating that commercial-grade fingerprinting can reliably single out a large fraction of users in practice.

Notably, cohort behavior does not consistently track user uniqueness. For *Akamai*, macOS and Linux exhibit nearly identical uniqueness rates (35.7% and 37.1%), yet their Simpson Indices differ by a factor of 3.5×: 135.48($\times 10^{-7}$) on macOS versus 478.27 on Linux.

Because the Simpson Index directly measures the probability that two randomly selected users share a fingerprint, this difference corresponds to a concrete change in likelihood: approximately 1 in 73,809 on macOS versus 1 in 20,909 on Linux. In other words, two randomly selected users are 3.5× more likely to share a fingerprint on Linux than on macOS.

This reflects a fundamental difference in cohort structure. While a similar fraction of users are unique on both platforms, the remaining non-unique users are distributed differently: on macOS they are spread across many smaller cohorts, whereas on Linux they are concentrated into fewer, larger cohorts. These results demonstrate that uniqueness alone is insufficient to characterize fingerprinting risk, as similar uniqueness rates can mask substantial differences in how users are grouped.

Key Insights: At real-world scale, a large fraction of users remain uniquely identifiable, confirming the effectiveness of modern fingerprinting techniques in practice and highlighting the impact of characteristic selection. However, uniqueness alone does not fully capture fingerprinting risk. Cohort-based analysis reveals that users deemed non-unique can be distributed very differently, either across many small cohorts or concentrated into a few large ones, leading to substantially different privacy implications. Consequently, **evaluating fingerprinting requires jointly considering both uniqueness and cohort structure**, as similar uniqueness rates can mask meaningful differences in how easily users can be re-identified in practice.

3.5 Measurement Contribution to Uniqueness

We analyze the data from multiple perspectives to identify high-impact mitigation targets to develop effective fingerprinting defenses.

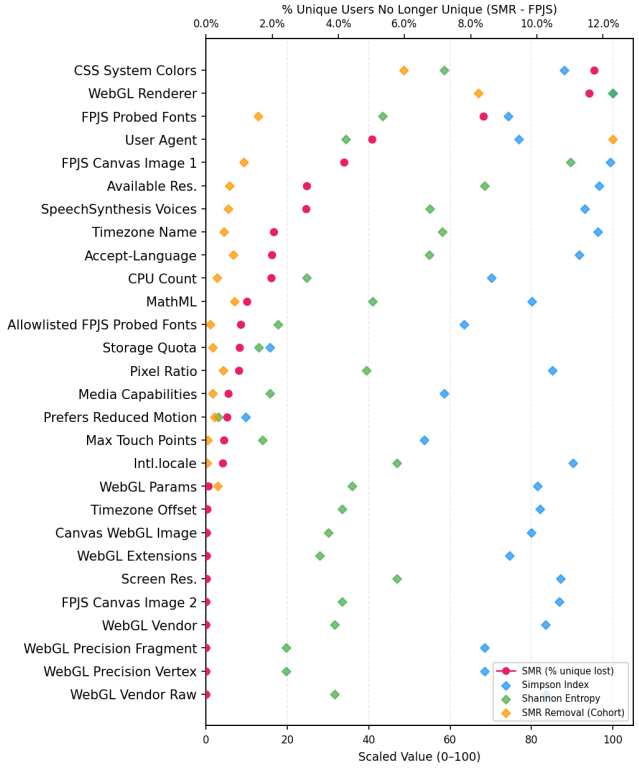


Figure 2: Individual measurement impact on fingerprint uniqueness across complementary metrics (Simpson Index, Shannon Entropy, and Single Measurement Removal (SMR)).

Methodology: To evaluate the distribution of individual measurements, we analyze each using three complementary techniques: **(1) Simpson Index (Sim):** measures concentration by estimating the probability that two users share the same value. **(2) Shannon Entropy:** captures the average information content of a measurement. **(3) Single Measurement Removal (SMR):** quantifies the marginal contribution of a measurement by removing it and observing the impact on unique users and cohort score.

While the Simpson Index and Shannon entropy characterize value distributions, SMR captures the direct impact of a measurement on fingerprint uniqueness.

Measurement Impact Analysis: As illustrated in Figure 2, each row represents a single measurement evaluated using our complementary techniques and ranked by its Single Measurement Removal (SMR) impact under the *Fingerprint Pro* model, aggregated across all OSes.

The top x-axis shows the SMR effect, i.e., the percentage of users that cease to be unique when the measurement is removed. The bottom x-axis reports the inverted Simpson Index, Shannon entropy, and SMR cohort improvement, each normalized to a 0–100 scale relative to their maximum values. The Simpson Index is inverted (`1 - val`) so that higher values indicate greater dispersion across the population, while higher Shannon entropy reflects greater identifying information. SMR cohort improvement captures the increase in cohort score when a measurement is removed, indicating its contribution to fragmenting users. Measurements are included if their SMR impact is $\geq 0.5\%$ or if any normalized metric is ≥ 65.

Limited Impact of Individual Measurements: Using SMR, we find that removing a single high-impact measurement when all others are retained can eliminate uniqueness for 11.7% of previously unique users. However, this effect drops off rapidly: only 8 measurements reduce uniqueness by more than 2%, indicating that no single measurement is critical to the overall effectiveness of a fingerprinter. Despite many measurements exhibiting high entropy or Simpson Index values (e.g., `Screen Resolution`), their individual contribution to uniqueness remains limited.

Key Insights: While removing any one of a small number of measurements reduces uniqueness by ≥5%, these are clear outliers: fingerprinting is generally not driven by any single dominant measurement. Moreover, commonly used statistical measures such as Shannon entropy or Simpson Index do not consistently predict a measurement's impact on uniqueness when considered individually.

Instead, most measurements contribute only marginally on their own, and fingerprinting effectiveness largely arises from the interaction of multiple, partially independent measurements. **Effective defenses should therefore prioritize combinations of high-impact measurements, rather than optimizing them in isolation**.

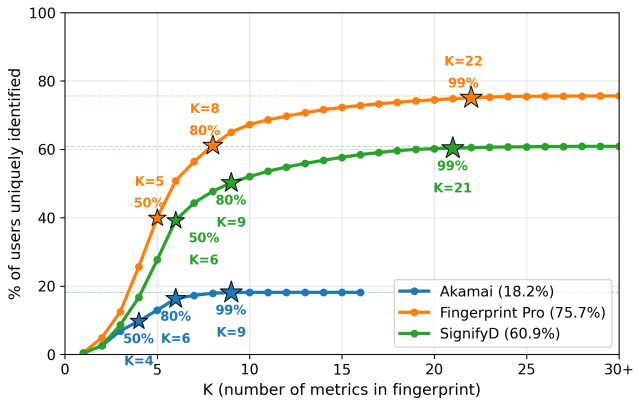


Figure 3: Cumulative coverage of unique users under incremental fingerprint construction.

3.6 Incremental Fingerprint Construction

Given that no individual measurement is crucial, we ask a complementary question: how many measurements are required in an optimal scenario to identify nearly all of the unique users that can be identified?

Methodology: Starting from an empty fingerprint, we greedily add measurements one at a time. At each step, we select the measurement whose addition maximizes the number of newly unique users, compute fingerprints using only the measurements selected so far, and record the cumulative number of unique users. This process continues until all measurements in the fingerprinter's set have been included. We perform this analysis both overall and per operating system using the same dataset.

Individual Attributes: Figure 3 shows the cumulative coverage curves for our three modeled fingerprinters' measurements (per-OS variants in Figure 6, Appendix H). The curves reveal that a small number of measurements produce disproportionately large gains when added (Appendix E explains possible causes for selected measurements).

For example, adding `Timezone Name` at position 4 adds 2.55 million unique users on Windows (*Fingerprint Pro*), while `User Agent` adds 1.07 million on Android (*Signifyd*). These measurements are not necessarily dominant in isolation (cf. Section 3.5), but they interact strongly with previously selected measurements, creating many newly unique combinations.

It is noteworthy that the greedy ordering differs substantially across platforms. On Windows, fonts, WebGL renderer, and canvas dominate early positions; on Android, locale and CSS colors lead; on Linux and macOS, timezone and screen availability are among the top measurements as shown in Table 10 in Appendix H.

Concentration of Fingerprinting Power: The shape of the curves directly addresses whether fingerprintability arises from aggregating many weak measurements or a concentrated set of strong ones. Rather than increasing linearly, the curves are sharply concave: across all three fingerprinters, no more than 10 measurements suffice to reach ~80% of unique users, while on a per-OS basis at least half of all measurements contribute less than 1%.

However, this concentration is not extreme. While a few measurements have relatively disproportionate impact, as noted above, even the most powerful individual measurement identifies fewer than 1.2% of users, and reaching 99% coverage requires 7–21 measurements depending on OS. Thus, identifying power is concentrated in a small set of interacting measurements rather than a few dominant attributes. This result highlights that fingerprinting effectiveness emerges from combinations of measurements rather than any single dominant high-entropy feature.

Key Insights: Fingerprint **uniqueness is concentrated in a small subset of measurements: no more than 10 suffice to identify 80% of unique users**, while a majority contribute negligible additional signal. Although the specific high-impact measurements vary by platform, the overall pattern—rapid initial gains

followed by diminishing returns—is consistent across all three fingerprinters. As a result, fingerprintability is driven by a concentrated set of interacting measurements rather than the accumulation of many independent measurements.

3.7 From Measurement to Defenses

The preceding analyses provide a direct basis for prioritizing fingerprinting defenses. Sections 3.5 and 3.6 show that fingerprintability is driven by a concentrated set of interacting measurements, and that high entropy alone does not reliably predict a measurement's contribution to uniqueness. Accordingly, rather than attempting to eliminate all sources of entropy, we prioritize measurements whose reduction or standardization has the greatest impact on uniqueness, while accounting for interactions between measurements and differences across platforms.

We use these findings to guide the mitigations evaluated in Section 4, subject to the web compatibility constraints described in Section 2.4. Section 4 presents these protections according to Firefox's deployment timeline: Phases 0–1 capture earlier experience-driven mitigations, while Phases 2–4 apply the data-driven approach developed from our measurement study. This distinction allows us to evaluate how progressively targeting the measurements identified by our analysis reduces fingerprintability in practice.

4 Real-World Mitigation

Section 3 showed that fingerprintability is driven by a relatively small set of interacting measurements whose importance varies across platforms. We now translate these findings into practical defenses by targeting high-impact measurements, while balancing the resulting privacy gains against web compatibility constraints.

This allows us to address the central question of our analysis: whether population-scale measurements can identify practical mitigation targets whose combined application materially reduces fingerprintability. Evaluated against simulated commercial fingerprinting services, these techniques reduce user uniqueness from 75% to 46% while preserving web compatibility.

We first situate our data-driven mitigations within Firefox's existing fingerprinting protections and deployment history, before evaluating their impact on user uniqueness and cohort structure.

Background on Fingerprinting Protection in Firefox: Firefox has introduced multiple mechanisms over time to limit information exposure and tracking on the web. Many of these mechanisms fall under the umbrella of *Enhanced Tracking Protection (ETP)* [79], which encompasses privacy protections including script blocking, storage isolation, and fingerprinting defenses.

Some of these protections are enabled by default in all browsing modes, while others are activated by default only in *Private Browsing Mode (PBM)* [92]. PBM applies behaviors associated with *Strict Enhanced Tracking Protection (ETP Strict)*. Users may also enable *ETP Strict* globally to apply these protections to regular browsing sessions.

4.1 Fingerprinting Protection Techniques

Section 3 showed that fingerprinting is driven by a relatively small set of high-impact, interacting measurements that account for a disproportionate share of user uniqueness at scale. Building on this insight, we use our simulation platform and large-scale dataset to systematically evaluate fingerprinting protection techniques, quantifying each technique's impact on user uniqueness and cohort score across five phases (cf. Table 2):

Phase 0 consists of ad hoc and best-effort mitigations, without a unified anti-fingerprinting strategy. **Phase 1** represents the first coordinated effort, guided by experience-driven tech leads rather than data. **Phase 2** introduces data-driven techniques derived from our initial measurement study, tempered by web compatibility reports from Phase 1 deployments (cf. Section 2.4); the resulting design changes are noted with each technique below. **Phase 3** expands on this approach with more aggressive mitigations informed by our second round of analysis as well as generative AI-assisted reverse engineering of sites using the affected APIs. These protections are under evaluation. **Phase 4** outlines further, forward-looking mitigations that push the boundary of entropy reduction while carefully considering their feasibility in a production browser.

FPT Phase 0 (Pre-2023)

CPU Core Count (Experience-Driven): Modern CPUs expose multiple logical processors, reported to websites via `navigator.hardwareConcurrency` [89] which has been identified as a fingerprinting vector as early as 2018 [108, 115]. Our data confirms this, as we observed 99 distinct values ranging from 1 to 768. Since 2016, Firefox has capped this value: 16 until 2021, then 128, disguising long-tail users.

WebGL Renderer String: WebGL (Web Graphics Library) is a JavaScript API for rendering 2D and 3D graphics [100] that, via `WEBGL_debug_renderer_info`, can reveal graphics hardware details [34]. Since 2021, Firefox mitigates this by returning a sanitized renderer string (e.g., "Apple M1, or similar") to reduce entropy.

Patch Version: The browser version string follows the format `major.minor.patch` and is exposed via the `User-Agent` header and through `navigator.userAgent`. Since patch updates introduce no web-visible changes, Firefox has omitted the patch number since 2012 (e.g., `143.0.1` → `143.0`) to reduce fingerprinting entropy.

FPT Phase 1 (2023-2024)

Fonts: Fonts are a major source of fingerprinting entropy [64, 139], as scripts can detect installed fonts via rendering differences. Phase 1 mitigates this by restricting websites to standardized, platform-specific font lists. This approach introduced user-reported regressions [74], which we addressed by restoring some affected fonts to the allowlist.

Math Functions: OS-specific implementations of trigonometric functions (e.g., `Math.sin,cos,tan` [86]) introduce fingerprinting entropy [114]. Firefox mitigates this by using the fdlibm library [43] to standardize results.

Web Audio: The Web Audio API [99] can be used for fingerprinting via system-specific processing differences [34]. Firefox mitigates

Fingerprinting Technique	*Technique Description*	*Fingerprinting Protection Technique*
FPT Phase 0 (Pre-2023 - Experience-Driven and Best Effort)		Firefox 16, 51, 87 (Deployed)
CPU Core Count (Experience-Driven)	returns the logical core count	cap the result to 128
WebGL Renderer String	returns raw renderer string of graphics driver	report sanitized string of graphic card driver
Patch Version	returns full version string of the browser	reports the patch version as .0
FPT Phase 1 (2023-2024 - Experience-Driven and Best Effort)		Firefox 118 (Deployed)
Fonts	exposes installed fonts	limit fonts to a reduced, specific list
Math Functions	math operations expose decimal differences	force all calculations inside math library
Web Audio	audio operations expose data differences	force all calculations inside math library
Canvas	allows drawing images using JS	add small bits of randomness into the image
FPT Phase 2 (2025 - Data-Driven Analysis)		Firefox 143 (Deployed)
CPU Core Count (Data-Driven)	returns the logical core count	report 4 if logical core count ≤ 4; else 8
Available Screen Resolution	returns available screen resolution	report screen resolution minus fixed offset
Font Updates	exposes installed fonts	exposes some fonts only in certain locales
FPT Phase 3 (2025-2026 - Data-Driven Analysis)		Firefox 157 (Experimental)
WebGL Vendor and Renderer	returns details of graphics hardware & software	reports constant values
Storage Quota	returns approximate space available to an origin	reports constant value
FPT Phase 4 (2026 and Beyond - Data-Driven Analysis)		(Under Consideration)
Software-Rendered Canvas	canvas draws images using dedicated hardware	canvas uses software backend to render
Accept-Language	exposes a list of locales a user accepts	report only first of list, and collapse locales
Timezone	returns timezone	collapse functionally identical timezones

Table 2: Fingerprinting Techniques and proposed Fingerprinting Protection Techniques (FPT).

this by standardizing computations using fdlibm, producing uniform outputs across users.

Canvas: Canvas fingerprinting uses the HTML5 `<canvas>` element [97] to derive device identifiers by rendering hidden content and exploiting subtle output differences across systems, extracted via APIs such as `.toDataURL()` [33]. Firefox mitigates this by deploying Canvas Randomization, injecting deterministic, context-dependent noise into outputs. This approach preserves stability within a domain and session while preventing cross-site reuse. Although limited, the injected noise did still result in reports of unexpected behavior and broken websites [75]; for example joining Google Meet calls failed until we exempted uniformly colored canvases from randomization [72]. Unlike other mitigations discussed in this paper, canvas randomization does not reduce exposed entropy but instead disrupts fingerprinting attempts. As a result, its effectiveness cannot be directly quantified through uniqueness metrics; we discuss its benefits in Section 5.

FPT Phase 2 (2025)

CPU Core Count (Data-Driven): Our empirical analysis showed that Phase 0 did not sufficiently reduce entropy from CPU core counts. In Phase 2, we cap values at 8 ($\geq$ 8 cores) or 4 ($<$ 8) to balance privacy and compatibility. Originally a constant value of 4 was chosen, but this degraded video encoding quality on Google Meet [76], motivating the two-value design above.

Available Screen Resolution: The two read-only screen interfaces `screen.availHeight,availWidth` [94] reveal screen resolution minus OS UI elements. We standardize these values using fixed offsets to prevent custom interface configurations from contributing to fingerprinting. On Windows, 48px is the dominant value at 51% of submissions (P96); on macOS, 76px is the 4th most common at 4.7% (P50); on Linux, 0px dominates at 58% (vs 10% for the next-most-common 40px).

Font Updates: We analyzed missing fonts from the Phase 1 allowlist and identified rarely installed ones, selecting nine also used by *Fingerprint Pro*. As these fonts are primarily used in specific locales, we proposed exposing them only when the corresponding locale is set, preserving compatibility while reducing entropy.

FPT Phase 3 (2025-2026)

WebGL Vendor and Renderer: Building on the WebGL renderer sanitization from Phase 0, both the renderer and the associated vendor string (identifying the graphics manufacturer) can be replaced with standardized values, e.g., `Mozilla`. Graphics-heavy sites that select rendering paths from these values are exempted, following static analysis of website JavaScript [77].

Storage Quota: The standard async method `estimate()` of the `StorageManager` [95] exposes the amount of storage available to a website for local data. Because websites should only use this as an estimate rather than a guarantee, we can normalize it to 10GB, which is reported by 91.82% of users.

FPT Phase 4 (2026 and Beyond)

Software-Rendered Canvas: Some canvas entropy arises from graphics hardware differences [33]. Using software rendering reduces this variability, offering privacy gains with low compatibility risk.

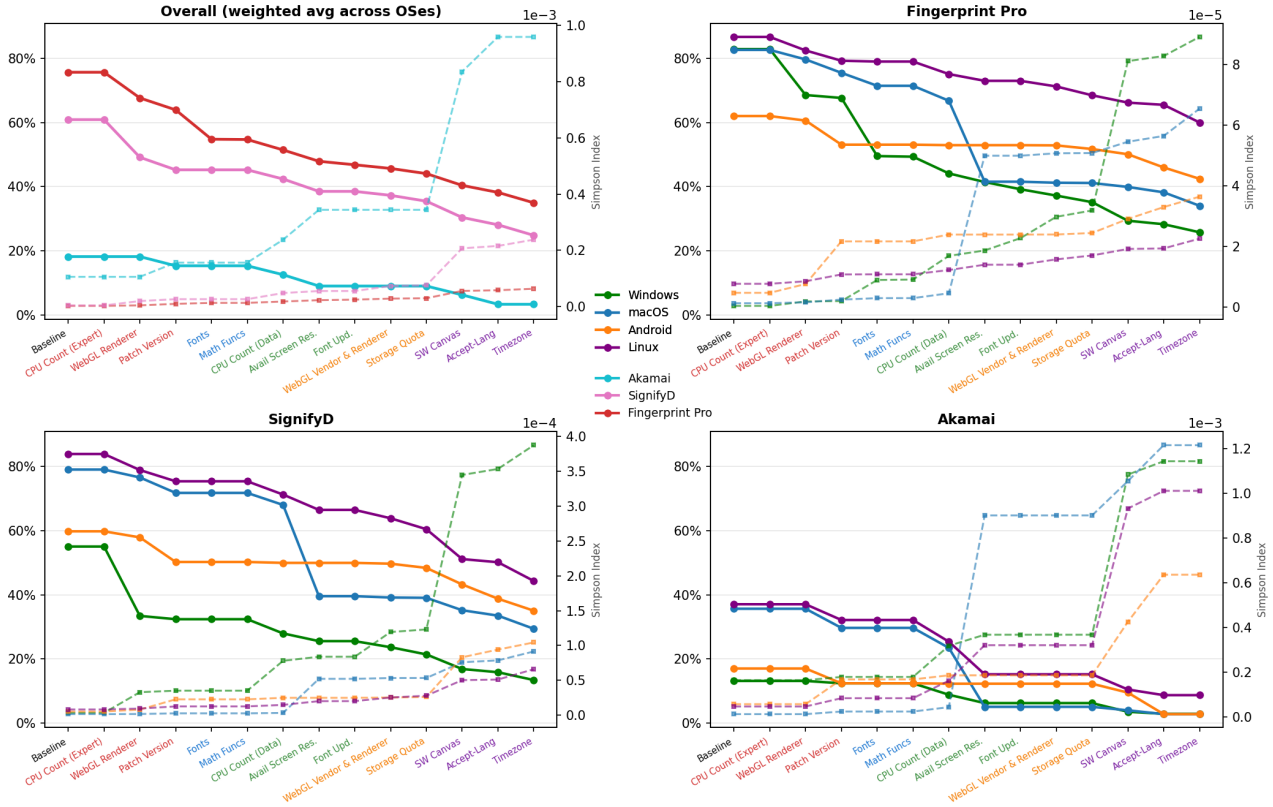


Figure 4: Impact of fingerprinting protections on user uniqueness and cohort score.

Accept-Language: The `Accept-Language` header exposes user language preferences, contributing significant entropy while remaining essential for compatibility. We evaluate reducing it by sending only the primary locale and collapsing regional variants (e.g., `en-US` → `en`). However, this approach remains under consideration due to compatibility risks, such as serving content in unintended languages.

Timezone: Web APIs expose UTC offset as well as timezone name (e.g., `America/New_York`), with the latter adding significant entropy (e.g., near-identical zones having distinct names). We evaluate collapsing identifiers per UTC offset, reducing entropy but introducing compatibility risks (e.g., DST discrepancies). This mitigation remains under consideration as a high-impact candidate, with ongoing evaluation of compatibility implications.

4.2 Effectiveness of Protections in Practice

Now, we evaluate the cumulative impact of our fingerprinting protection techniques by analyzing their impact on user uniqueness through the lens of three real-world fingerprinting services (*Fingerprint Pro*, *Signifyd*, and *Akamai*) over our dataset. Figure 4 reports the fraction of users classified as unique (solid lines) and the corresponding cohort score (Simpson Index, dashed lines) for each protection technique. The per-protection numbers are tabulated in Appendix D.

Following the initial "experience-driven" phases, the data-driven mitigations introduced in Phases 2 and 3 steadily reduce the fraction of unique users for *Fingerprint Pro* from approximately 75% to 44%, while increasing the Simpson Index by an order of magnitude (weighted average across operating systems for *Fingerprint Pro*; Table 3 in Appendix D). Similar trends hold for *Signifyd*, where uniqueness falls from 60.9% to 35.4% over the same phases with a comparable rise in the Simpson Index. Furthermore, ongoing work in Phase 4 aims to further reduce uniqueness by an additional ~10 percentage points.

These improvements come unevenly: only 6 out of 64 scenarios (8 protection techniques from Phase 2 onward × 4 OSes × 2 fingerprinters[2]) reduce uniqueness by more than 15% relative to the prior protection. Among these, Software Canvas accounts for three (both *Fingerprint Pro* and *Signifyd* for Windows, and *Signifyd* for Linux), Available Screen Resolution for two (macOS for both fingerprinters), and Timezone for one (*Signifyd* for Windows). Appendix E explores selected outliers. While only half of scenarios exceed 5% relative improvement, every protection technique reduces uniqueness by at least 5% for some OS/fingerprinter combination.

Together, these protection techniques drive down uniqueness nearly 30 percentage points, but examined individually (Figure 5) the picture is different. For example, consider the discrepancy in

[2]Because *Akamai* collects fewer metrics, it is more affected by individual protection techniques, so we focus on the two fingerprinters with larger measurement sets.

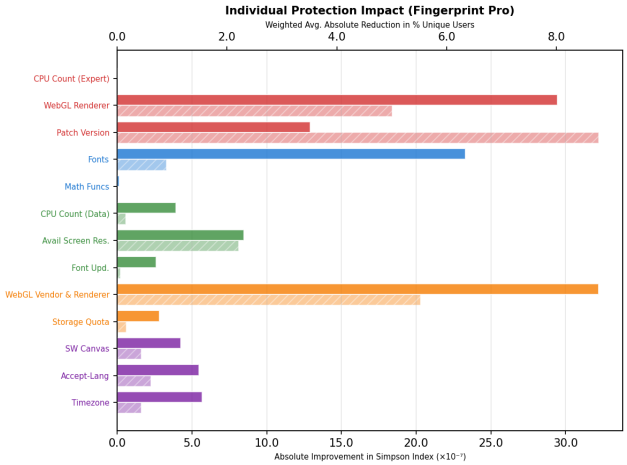


Figure 5: Impact of each fingerprinting protection technique applied in isolation against the baseline.

the WebGL Vendor and Renderer protection. It reports an 8.8 percentage point improvement when applied individually (Table 4 in Appendix D), but only a 1.1 point improvement when applied cumulatively (Table 3 in Appendix D). This is explained by redundancy in the techniques: the earlier WebGL Renderer protection (which sanitizes the Renderer string) subsumes most of the gains of the later one, which sets the string to a constant value.

However, the opposite can also be true: on Windows for *Fingerprint Pro*, Software Canvas reports only a 1.3 percentage point improvement individually (Table 6 in Appendix D), but a 5.8 point improvement cumulatively (Table 5 in Appendix D). Its cohort score tells a similar story: individually, Software Canvas barely moves the score (a 0.3×10^{-7} increase from baseline, cf. Table 1). Cumulatively, it jumps from 318.2 to 811.5×10^{-7} (a 2.6× increase), driven by prior protections that have already concentrated non-unique users into smaller cohorts.

Indeed, browser features can be correlated [17], and our work confirms that interaction effects between protections amplify modest individual contributions and reinforces that OS segmentation is essential for designing protections and evaluating their impact. Importantly, these gains are achieved while maintaining web compatibility.

We deliberately avoid simulating adaptive adversaries, as doing so would require subjective assumptions about feature selection and matching strategies, and risk inadvertently designing stronger fingerprinting techniques. Instead, we evaluate robustness across feature subsets and optimal selection bounds, which provide attacker-agnostic guarantees.

Overall, these results demonstrate that data-driven, deployable defenses significantly weaken real-world fingerprinting systems at scale, reducing user uniqueness from 75% to 46%.

The reduction we report is specific to Firefox and its user population, and the specific numbers would not transfer directly to other browsers—particularly those built on different engines. Other engines expose Web APIs that Firefox does not, enlarging the fingerprinting surface: Chromium, for example, implements the read-only property `navigator.deviceMemory` as well as the API method `navigator.userAgentData.getHighEntropyValues()`. Internal implementations of APIs such as canvas and Web Audio also differ, producing different values and therefore different baseline entropy, and uniqueness depends further on population composition, which differs across browsers.

The mitigations themselves, however, are not Firefox-specific: each targets a standardized Web API, reduces the information that API exposes, and was designed against real web compatibility constraints. We therefore expect the same techniques to apply to other major browsers (e.g., Chrome, Safari, Edge, Brave; cf. Section 6.4), though the magnitude of the benefit would need to be measured in each.

5 Design Considerations

Building on the deployed protections of Section 4, we step back to situate our approach among other anti-fingerprinting strategies and outline remaining challenges. We contrast it with two widely studied alternatives—randomization and targeted disruptions—and then consider the open problem of sustaining these gains as fingerprinting providers adapt.

5.1 Measurement-Driven Design

Our results yield five principles that depart from prior work. **(1) Select targets by removal, not entropy:** Shannon entropy and the Simpson Index do not reliably predict a measurement's contribution to uniqueness (Figure 2). **(2) Report cohort structure, not uniqueness alone:** For *Akamai*, macOS and Linux differ by 1.4 percentage points in uniqueness but 3.5× in Simpson Index (Section 3.4): two random Linux users share a fingerprint far more readily. **(3) Spend compatibility risk on the head, not the tail:** ten measurements suffice to identify 80% of unique users (Section 3.6); the remainder rarely justify breakage risk. **(4) Evaluate mitigations jointly:** software-rendered canvas gains 1.3 percentage points alone but 5.8 once other protections have concentrated cohorts (Windows, *Fingerprint Pro*; Tables 6 and 5 in Appendix D). Assessed in isolation, such mitigations appear not to warrant their cost. **(5) Segment by operating system:** high-impact measurements vary by platform, so a single global policy is mis-targeted for most users.

5.2 Alternative Anti-Fingerprinting Approaches

Our work demonstrates that standardizing the values web APIs return can substantially reduce browser fingerprintability at scale. We briefly contrast this approach with two alternatives — randomization and targeted disruptions — and discuss where each fits.

Randomization: Fingerprinting techniques rely on web APIs returning stable and consistent values across time and contexts. Introducing randomness violates this assumption, thereby disrupting fingerprinting algorithms and reducing their ability to reliably re-identify users. Randomization-based defenses have been widely studied and are often detected or bypassed [107], yet they remain valuable; Brave applies them the most broadly of any major browser

(cf. Section 6.4). While their efficacy is difficult to measure without access to proprietary algorithms, they still provide benefits. Fingerprinting providers adapt at different speeds, and randomization increases the cost of recovering underlying signals, either through added engineering effort or performance overhead. It is also possible that sophisticated systems may detect and discard randomized measurements altogether [41, 67]. Consequently, randomization can provide both short- and long-term privacy benefits, though it may introduce compatibility issues and can be bypassed by determined adversaries.

Targeted Disruptions: Another approach is to identify fingerprinting attempts and selectively block or disrupt them. One method is to block network requests retrieving fingerprinting scripts by matching known URLs (e.g., through crowd-sourced blocklists). However, recent work [66] has shown that blocklist-based defenses are ineffective for a large fraction of fingerprinting scripts as trackers can adopt evasion techniques such as CNAME cloaking [32] or simply rotate the script URL. Another method is to disrupt only the fingerprinting script on a webpage, without disrupting the rest of the page's operation. By detecting the fingerprinting script using heuristics (typically static patterns), specific operations can be disrupted (e.g. by adding noise only to the fingerprinting script's canvas operations). Safari deploys this approach, applying canvas noise outside Private Browsing only to scripts on a list distributed to the browser at runtime (cf. Section 6.4). Both these approaches are similarly prone to evasion, as they rely on static patterns, and as with randomization, fingerprinting providers can adapt to such defenses with varying effort and speed. Further, both script blocking and targeted API restriction can cause significant website breakage [54].

Combining the Three: Ultimately, these approaches are complementary rather than mutually exclusive. Standardization provides significant and measurable improvements, but is limited by both web compatibility constraints and when users have uncommon hardware (e.g., extremely large monitors) or unusual settings (e.g., the Mandarin-localized visitor to a local Icelandic website from before). Randomization fills that gap, accepting some compatibility risk and the possibility of bypass in exchange for frustrating tracking attempts by at least some fingerprinting services for a period of time. Targeted disruptions then add a final layer by blocking scripts known to perform fingerprinting outright, before they get to execute at all. Firefox's configuration reflects this layering: standardization across Phases 0–4, Canvas Randomization for irreducible canvas output, and Enhanced Tracking Protection blocklists for known fingerprinting URLs. An effective defense strategy should combine all three.

5.3 Outlook

Our results show that standardization informed by real-world population measurements can substantially reduce browser fingerprintability while preserving web compatibility. By aligning exposed characteristics with common values, browsers concentrate users into larger cohorts and mask atypical configurations, achieving meaningful privacy gains with fewer compatibility regressions than more comprehensive approaches such as Tor Browser or RFP mode. Since Firefox 115, RFP has generated 153 breakage reports compared to FPP's 59 over the same period, despite being used by at least three orders of magnitude fewer users [81].

Sustaining these gains, however, requires continued measurement. Even within the span of this study, we observed *Fingerprint Pro* and *Akamai* expand the measurements they collect, illustrating the continuing adaptation between fingerprinters and browser defenses. The cat-and-mouse dynamic between browsers and fingerprinters means ongoing measurement remains essential, and future work must continue to track how fingerprinters react to the defenses proposed here. Our measurement-driven approach provides a basis for responding to this evolution: re-evaluating which measurements drive uniqueness as fingerprinting techniques change, and using these results to prioritize subsequent protections without unnecessarily sacrificing functionality. The data underpinning our evaluation will empower the next generation of fingerprinting protections, and the reduction in user uniqueness we have demonstrated limits the exposure of user characteristics without sacrificing functionality.

6 Related Work

Since Eckersley's 2010 work demonstrating browser fingerprinting [33], the technique has become ubiquitous and a threat to user privacy. Numerous studies have documented the prevalence and sophistication of both cookie-based and fingerprinting-based tracking. In this section, we summarize the state of the art and position our work within it.

6.1 Cookie-Based Tracking

Early work by Acar et al. and Englehardt et al. demonstrated that cookie-based tracking is pervasive in practice [1, 34, 35]. Subsequent studies confirmed that trackers exploit cookies and other stateful identifiers to monitor users across sites [6, 19, 26, 29, 31, 32, 38, 55, 68, 69, 105, 128]. In response, browser vendors introduced countermeasures against cookie-based tracking [14, 82, 112]. As these protections became widespread, tracking increasingly shifted toward alternative vectors, most notably browser fingerprinting.

6.2 Fingerprinting-Based Tracking

We group prior work into three categories: (a) large-scale measurement studies, (b) analyses of high-entropy characteristics, and (c) evaluations of fingerprinting effectiveness. Together, these strands of research characterize both the prevalence of fingerprinting and the mechanisms that enable it, but often in isolation or at limited scale.

a) Large-Scale Measurement Studies: Prior work has measured in-the-wild fingerprinting for tracking [1, 34, 54, 66] and authentication [116]. Notably, Iqbal et al. [54] used learning-based techniques to detect previously unknown fingerprinting scripts without relying on known fingerprinting APIs. Other studies examine user fingerprints themselves: Wu et al. [138] analyze attacker-crafted fingerprints from commercial sites, while Pau et al. [110] provide a small open dataset.

b) High-Entropy Fingerprinting Characteristics: A second line of work identifies signals that drive fingerprintability. Su et al. [119] and Nikkhah Bahrami et al. [108] detect emerging fingerprinting APIs using program analysis and machine learning. Other work demonstrates the strength of specific measurements, including CSS-based probes [64] and features such as fonts, plugins, and Canvas [60], while entropy analyses highlight window size, canvas rendering, and the speech engine [61]. Bacis et al. [17] further studied the entropy of *correlated* browser APIs. Together, these findings show that a small set of measurements contributes disproportionately to user uniqueness.

c) Fingerprinting Effectiveness: Prior work evaluates fingerprinting effectiveness in practice. Liu et al. [65] provide evidence of its use in tracking and advertising, while Gómez-Boix et al. [45] show that uniqueness decreases at scale. Longitudinal and cross-platform studies demonstrate persistence over time [113] and cross-browser tracking via OS- and hardware-level features [27]. Andriamilanto et al. [8] further show that fingerprints exhibit high uniqueness and stability. Together, these results highlight both the effectiveness and limitations of fingerprinting in real-world settings, particularly the gap between controlled measurements and deployment at scale.

6.3 Distinction from Prior Work

The most comprehensive prior studies rely on volunteer-based or website-specific datasets and are limited in scale, typically ranging from 120,000 to 2 million users. In contrast, we measure fingerprinting *in the wild* using data from nearly 25 million clients, representing a **12× to 208× increase in scale** over prior work. This enables a more representative and statistically robust assessment of fingerprint uniqueness and cohort score. Moreover, rather than relying on synthetic or researcher-curated datasets, our analysis uses **real-world telemetry data**, capturing actual user configurations at population scale. By modeling commercial fingerprinting services and deploying mitigations in Firefox, a production browser, we **bridge the gap between measurement and practical defense**.

6.4 Fingerprinting Defenses in Other Browsers

Other browsers have put significant effort into fingerprinting defenses over the years as well, although typically added API by API and without public measurement of their effect on uniqueness. Chrome has pursued several anti-fingerprinting workstreams: it abandoned its Privacy Budget proposal [50], which would have capped the total entropy a site could observe, while its proposals to reduce `Accept-Language` entropy [49] and to noise canvas readbacks [47] remain unshipped as of August 2026. Safari excludes user-installed fonts and freezes the User-Agent against minor updates [135], and noises canvas, WebGL, and Web Audio [134]. It also returns a hard-coded image for one particular fingerprinting probe [133]. Brave, built on Chromium, has aggressively pursued randomness, described as 'farbling' in their documentation, in many APIs including canvas, Web Audio, WebGL, `hardwareConcurrency`, and `deviceMemory` [24, 25]. It also declines APIs that Chromium exposes and neither Firefox nor Safari implements [23], and recently standardized WebGL vendor and renderer strings while randomizing the extension list [22].

In all browsers, the scope of these protections varies. In Safari canvas noise applies to all scripts only in Private Browsing—elsewhere, only to scripts on a list distributed at runtime. In the past Brave also offered a stricter opt-in mode with more aggressive protections [21].

Tor Browser [122, 123] pursues uniformity across APIs [111], with canvas the exception: RFP substitutes randomized data for the true rendering, to frustrate linking even the very limited fingerprint Tor Browser affords [85]; Mullvad Browser packages the same hardening without the Tor network [104]. Browsers downstream, especially of Firefox in particular, inherit these protections and curate modifications to them: Phoenix maintains over 600 per-site overrides against Firefox's fingerprinting-protection targets, over 200 of which disable UTC reporting [28, 56]. LibreWolf has long shipped Firefox's Resist Fingerprinting mode, but is discussing migrating to Firefox's fingerprinting-protection framework for its per-domain overrides [63].

Individual APIs thus receive markedly different treatment—`hardwareConcurrency` is capped by Firefox, randomized by Safari and Brave, and unmodified by Chrome. Guidance for specification authors exists [130], but does not establish which characteristics warrant intervention, or in what order.

Our work complements these defenses by providing population-scale measurements for identifying and prioritizing high-impact fingerprinting surfaces.

7 Conclusion

To the best of our knowledge, we present the first large-scale, real-world study of browser fingerprinting based on telemetry data from nearly 25 million Firefox users. At this scale, we show that fingerprint uniqueness is highly sensitive to dataset size and remains structurally significant: approximately 75% of users are uniquely identifiable by commercial-grade fingerprinting services without protections.

Overall, our findings demonstrate that browser fingerprinting is not an unavoidable property of the web, but can be substantially reduced at scale. Identifiability is driven by a relatively small set of high-impact, interacting features, rather than the accumulation of many independent signals. This concentration enables effective mitigation through a data-driven, privacy-through-conformity strategy that enlarges anonymity sets while preserving functionality. Beyond measurement, we translate these insights into deployable defenses. By simulating real-world fingerprinting services, we evaluate concrete browser engineering decisions and quantify their privacy impact. The resulting protections, shipped in Firefox 143, reduce uniqueness from 75% to 46% while maintaining web compatibility.

References

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juárez, Arvind Narayanan, and Claudia Díaz. 2014. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Conference on Computer and Communications Security*. ACM, New York, NY, USA.

[2] GitHub account: aframevr. 2021. A-Frame: Camera movement broken with resistFingerprinting. https://github.com/aframevr/aframe/issues/4793 (accessed: September, 2026).

[3] GitHub account: philc. 2023. Vimium: Alt+number shortcuts broken by resistFingerprinting. https://github.com/philc/vimium/issues/4053 (accessed: September, 2026).

[4] GitHub account: ppeccin. 2020. WebMSX: Performance degradation with resistFingerprinting. https://github.com/ppeccin/WebMSX/issues/32 (accessed: September, 2026).

[5] GitHub account: rstudio. 2019. RStudio: Keyboard shortcuts mangled by resistFingerprinting. https://github.com/rstudio/rstudio/issues/4240 (accessed: September, 2026).

[6] Zubair Ahmad, Samuele Casarin, and Stefano Calzavara. 2023. An Empirical Analysis of Web Storage and Its Applications to Web Tracking. In *Transactions on the Web*. ACM, New York, NY, USA.

[7] Akamai. 2026. Bot Manager. https://www.akamai.com/products/bot-manager (accessed: September, 2026).

[8] Nampoina Andriamilanto, T. Allard, and Gaëtan Le Guelvouit. 2020. "Guess Who?" Large-Scale Data-Centric Study of the Adequacy of Browser Fingerprints for Web Authentication. In *Transactions on the Web*. Springer, Cham, Switzerland.

[9] Anthropic. 2026. The best AI for developers. https://claude.com/solutions/coding (accessed: September, 2026).

[10] Apache. 2018. Firefox about:config setting privacy.resistFingerprinting=true disables ctrl+alt+shift menu. Apache JIRA. https://issues.apache.org/jira/browse/GUACAMOLE-554 (accessed: September, 2026).

[11] Apple. 2022. Challenge: Private Access Tokens. https://developer.apple.com/news/?id=huqjyh7k (accessed: September, 2026).

[12] Apple. 2026. Apple announces powerful new privacy and security features. https://www.apple.com/au/newsroom/2023/06/apple-announces-powerful-new-privacy-and-security-features/ (accessed: September, 2026).

[13] Apple. 2026. Configuration Profile Reference. https://developer.apple.com/business/documentation/Configuration-Profile-Reference.pdf (accessed: September, 2026).

[14] Apple. 2026. Intelligent Tracking Prevention. https://www.apple.com/privacy/features/ (accessed: September, 2026).

[15] Apple. 2026. Use Private Browsing windows. https://help.apple.com/safari/mac/8.0/en.lproj/ibrw1069.html (accessed: September, 2026).

[16] Mohammadreza Ashouri. 2019. A Large-Scale Analysis of Browser Fingerprinting via Chrome Instrumentation. In *Conference on Internet Monitoring and Protection*. IARIA, Nice, France.

[17] Enrico Bacis, Igor Bilogrevic, Robert Busa-Fekete, Asanka Herath, Antonio Sartori, and Umar Syed. 2024. Assessing Web Fingerprinting Risk. In *World Wide Web Conference*. ACM, New York, NY, USA.

[18] A. Barth. 2026. HTTP State Management Mechanism. https://datatracker.ietf.org/doc/html/rfc6265 (accessed: September, 2026).

[19] Nataliia Bielova. 2017. Web Tracking Technologies and Protection Mechanisms. In *Conference on Computer and Communications Security*. ACM, New York, NY, USA.

[20] Igor Bilogrevic, Balazs Engedy, Judson L. Porter III, Nina Taft, Kamila Hasanbega, Andrew Paseltiner, Hwi Kyoung Lee, Edward Jung, Meggyn Watkins, PJ McLachlan, and Jason James. 2021. "Shhh...be quiet!" Reducing the Unwanted Interruptions of Notification Permission Prompts on Chrome. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.

[21] Brave. 2026. Brave browser simplifies its fingerprinting protections. https://brave.com/privacy-updates/28-sunsetting-strict-fingerprinting-mode/ (accessed: September, 2026).

[22] Brave. 2026. Brave improves protections against GPU fingerprinting. https://brave.com/privacy-updates/38-webgl-webgpu-fingerprinting-protections/ (accessed: September, 2026).

[23] Brave. 2026. Deviations from Chromium (features we disable or remove). https://github.com/brave/brave-browser/wiki/Deviations-from-Chromium-(features-we-disable-or-remove) (accessed: September, 2026).

[24] Brave. 2026. Fingerprinting defenses 2.0. https://brave.com/privacy-updates/4-fingerprinting-defenses-2.0/ (accessed: September, 2026).

[25] Brave. 2026. Fingerprinting Protections. https://github.com/brave/brave-browser/wiki/Fingerprinting-Protections (accessed: September, 2026).

[26] Tomasz Bujlow, Valentín Carela-Español, Josep Solé-Pareta, and Pere Barlet-Ros. 2017. A Survey on Web Tracking: Mechanisms, Implications, and Defenses. In *Proceedings of the IEEE*. IEEE, Piscataway, NJ, USA.

[27] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *Network and Distributed System Security Symposium*. Internet Society, Reston, VA, USA.

[28] celenity. 2026. Phoenix. https://codeberg.org/celenity/Phoenix (accessed: September, 2026).

[29] Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. 2021. Cookie Swap Party: Abusing First-Party Cookies for Web Tracking. In *World Wide Web Conference*. ACM, New York, NY, USA.

[30] CreepJS. 2026. CreepJS - Real-time analysis of your browser's unique characteristics. https://creepjs.org/ (accessed: September, 2026).

[31] Savino Dambra, Iskander Sánchez-Rola, Leyla Bilge, and Davide Balzarotti. 2022. When Sally Met Trackers: Web Tracking From the Users' Perspective. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.

[32] Ha Dao and Kensuke Fukuda. 2020. Characterizing CNAME Cloaking-based Tracking on the Web. In *Traffic Monitoring and Analysis*. Springer, Cham, Switzerland.

[33] Peter Eckersley. 2010. How Unique is Your Web Browser?. In *Privacy Enhancing Technologies*. De Gruyter, Berlin, Germany.

[34] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Conference on Computer and Communications Security*. ACM, New York, NY, USA.

[35] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W. Felten. 2015. Cookies That Give You Away: The Surveillance Implications of Web Tracking. In *World Wide Web Conference*. ACM, New York, NY, USA.

[36] Julian Fietkau, Kashyap Thimmaraju, Felix Kybranz, Sebastian Neef, and Jean-Pierre Seifert. 2021. The Elephant in the Background: A Quantitative Approach to Empower Users Against Web Browser Fingerprinting. In *Workshop on Privacy in the Electronic Society*. ACM, New York, NY, USA.

[37] Fingerprint. 2026. Fingerprint Pro. https://fingerprint.com/ (accessed: September, 2026).

[38] Gertjan Franken, Tom van Goethem, and Wouter Joosen. 2018. Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.

[39] GitHub account: arkenfox. 2026. TorZillaPrint - A comprehensive, all-in-one, fingerprinting test suite. https://github.com/arkenfox/TZP (accessed: September, 2026).

[40] GitHub account: fingerprintjs. 2026. Content blockers. https://github.com/fingerprintjs/fingerprintjs/blob/30361297159ef130fe18635dae2a595a16083159/docs/content_blockers.md (accessed: September, 2026).

[41] GitHub account: fingerprintjs. 2026. FingerprintJS - Source Code Comment: 'Some browsers add a noise to the canvas'. https://github.com/fingerprintjs/fingerprintjs/blob/30361297159ef130fe18635dae2a595a16083159/src/sources/canvas.ts#L83 (accessed: September, 2026).

[42] GitHub account: fingerprintjs. 2026. Source Code: dom_blockers.ts. https://github.com/fingerprintjs/fingerprintjs/blob/master/src/sources/dom_blockers.ts (accessed: September, 2026).

[43] GitHub account: freemint. 2026. fdlibm Math Library. https://github.com/freemint/fdlibm (accessed: September, 2026).

[44] Alejandro Gómez-Boix, Davide Frey, Yérom-David Bromberg, and Benoit Baudry. 2019. A Collaborative Strategy for Mitigating Tracking through Browser Fingerprinting. In *Workshop on Moving Target Defense*. ACM, New York, NY, USA.

[45] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *World Wide Web Conference*. ACM, New York, NY, USA.

[46] Google. 2026. Material You. https://source.android.com/docs/core/display/material (accessed: September, 2026).

[47] Google. 2026. Noising canvas readbacks in Incognito. https://chromestatus.com/feature/5589949602332672 (accessed: September, 2026).

[48] Google. 2026. Open incognito tab. https://chromewebstore.google.com/detail/open-incognito-tab/ebgmlfdcgihhfheckfdmhnmedjigogmm?hl=en (accessed: September, 2026).

[49] Google. 2026. Reduce fingerprinting in the Accept-Language header. https://github.com/explainers-by-googlers/reduce-accept-language (accessed: October, 2025).

[50] Google. 2026. The Privacy Budget explainer. https://github.com/bslassey/privacy-budget (no longer active; accessed: September, 2026).

[51] Inria France (Team of Researchers). 2026. AmIUnique - Inform users about their position within the web ecosystem. https://amiunique.org/ (accessed: September, 2026).

[52] Internet Engineering Task Force (IETF). 2026. HTTP Strict Transport Security (HSTS). https://tools.ietf.org/html/rfc6797 (accessed: September, 2026).

[53] Internet Engineering Task Force (IETF). 2026. Hypertext Transfer Protocol – HTTP/1. https://tools.ietf.org/html/rfc2616 (accessed: September, 2026).

[54] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2020. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *Symposium on Security and Privacy*. IEEE, Piscataway, NJ, USA.

[55] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. 2018. AdGraph: A Graph-Based Approach to Ad and Tracker Blocking. In *Symposium on Security and Privacy*. IEEE, Piscataway, NJ, USA.

[56] IronFox OSS. 2026. IronFox: fingerprinting protection overrides. https://gitlab.com/ironfox-oss/IronFox/-/tree/dev/patches/gecko-overlay/ironfox/dumps (accessed: September, 2026).

[57] Dennis Jackson. 2026. PACT: Anonymous Credentials for the Web. https://hacks.mozilla.org/2026/06/pact-anonymous-credentials-for-the-web/ (accessed: September, 2026).

[58] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *Network and Distributed System Security Symposium*. Internet Society, Reston, VA, USA.

[59] Pierre Laperdrix, Nataliia Bielova, Benoît Baudry, and Gildas Avoine. 2019. Browser Fingerprinting: A survey. In *Transactions on the Web*. ACM, New York, NY, USA.

[60] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *Symposium on Security and Privacy*. IEEE, Piscataway, NJ, USA.

[61] Vicki Wei Qi Lee, Shih Yin Ooi, and Ying Han Pang. 2023. Assessing the Importance of Browser Fingerprint Attributes towards User Profiling through Clustering Algorithms. In *Symposium on Computer Applications & Industrial Electronics*. IEEE, Piscataway, NJ, USA.

[62] Song Li and Yinzhi Cao. 2020. Who Touched My Browser Fingerprint?: A Large-scale Measurement Study and Classification of Fingerprint Dynamics. In *Internet Measurement Conference*. ACM, New York, NY, USA.

[63] LibreWolf. 2026. Issue 2121: Move from RFP to FPP. https://codeberg.org/librewolf/issues/issues/2121 (accessed: September, 2026).

[64] 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 *Symposium on Security and Privacy*. IEEE, Piscataway, NJ, USA.

[65] Zengrui Liu, Jimmy Dani, Yinzhi Cao, Shujiang Wu, and Nitesh Saxena. 2025. The First Early Evidence of the Use of Browser Fingerprinting for Online Tracking. In *World Wide Web Conference*. ACM, New York, NY, USA.

[66] Elisa Luo, Tom Ritter, Stefan Savage, and Geoffrey M. Voelker. 2025. Canvassing the Fingerprinters: Characterizing Canvas Fingerprinting Use Across the Web. In *Internet Measurement Conference*. ACM, New York, NY, USA.

[67] Elisa Luo, Tom Ritter, Stefan Savage, and Geoffrey M. Voelker. 2026. Understanding Server-side Commercial Fingerprinting. In *World Wide Web Conference*. ACM, New York, NY, USA.

[68] Maryam Mehrnezhad, Kovila P. L. Coopamootoo, and Ehsan Toreini. 2021. How Can and Would People Protect From Online Tracking?. In *Privacy Enhancing Technologies*. De Gruyter, Berlin, Germany.

[69] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar R. Weippl. 2017. Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools. In *European Symposium on Security and Privacy*. IEEE, Piscataway, NJ, USA.

[70] Microsoft. 2026. Browse InPrivate in Microsoft Edge. https://support.microsoft.com/en-us/microsoft-edge/browse-inprivate-in-microsoft-edge-cd2c9a48-0bc4-b98e-5e46-ac40c84e27e2 (accessed: September, 2026).

[71] Cloistered Monkey. 2023. Mozilla Madness: Resist Fingerprinting. https://necromuralist.github.io/posts/mozilla-madness-resist-fingerprinting/ (accessed: September, 2026).

[72] Mozilla. 2023. Bug 1852541: meet.google.com - Cannot join meetings when Canvas Randomization is enabled. https://bugzilla.mozilla.org/show_bug.cgi?id=1852541 (accessed: September, 2026).

[73] Mozilla. 2026. Breakage from Fingerprinting Resistance. https://bugzilla.mozilla.org/show_bug.cgi?id=1507517 (accessed: September, 2026).

[74] Mozilla. 2026. Bug 1827475: Tracking bug for font visibility restrictions causing web compatibility issues. https://bugzilla.mozilla.org/show_bug.cgi?id=1827475 (accessed: September, 2026).

[75] Mozilla. 2026. Bug 1957426: Tracking bug for canvas randomization causing web compatibility issues. https://bugzilla.mozilla.org/show_bug.cgi?id=1957426 (accessed: September, 2026).

[76] Mozilla. 2026. Bug 1984132: Sent video in Google Meet is low quality. https://bugzilla.mozilla.org/show_bug.cgi?id=1984132 (accessed: September, 2026).

[77] Mozilla. 2026. Bug 2036879: Address Graphics-Heavy Website Vendor/Renderer detection. https://bugzilla.mozilla.org/show_bug.cgi?id=2036879 (accessed: September, 2026).

[78] Mozilla. 2026. Bug List - Breakage from Incorrectly Reporting the Operating System. https://bugzilla.mozilla.org/buglist.cgi?quicksearch=1509829%2C1610762%2C1755832%2C1826098%2C1840385%2C1519122%2C1405810%2C1404608%2C1409001%2C1441003%2C1621988%2C1696669%2C1509829&list_id=17938923 (accessed: September, 2026).

[79] Mozilla. 2026. Enhanced Tracking Protection in Firefox. https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop/ (accessed: September, 2026).

[80] Mozilla. 2026. Experimenter: Mozilla's Experimentation Platform. https://experimenter.info/ (accessed: September, 2026).

[81] Mozilla. 2026. Fingerprinting/Breakage. https://wiki.mozilla.org/Fingerprinting/Breakage (accessed: September, 2026).

[82] Mozilla. 2026. Firefox 86 Introduces Total Cookie Protection. https://blog.mozilla.org/security/2021/02/23/total-cookie-protection/ (accessed: September, 2026).

[83] Mozilla. 2026. Firefox Public Data Report. https://data.firefox.com/dashboard/usage-behavior (accessed: September, 2026).

[84] Mozilla. 2026. Firefox Source Tree Documentation - Code quality. https://firefox-source-docs.mozilla.org/code-quality/index.html (accessed: September, 2026).

[85] Mozilla. 2026. GeneratePlaceholderCanvasData.h. https://searchfox.org/firefox-main/source/dom/canvas/GeneratePlaceholderCanvasData.h (accessed: September, 2026).

[86] Mozilla. 2026. Math.sin(). https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/sin (accessed: September, 2026).

[87] Mozilla. 2026. Mozilla Data Collection. https://wiki.mozilla.org/Data_Collection (accessed: September, 2026).

[88] Mozilla. 2026. Mozilla Data Documentation. https://docs.telemetry.mozilla.org/ (accessed: September, 2026).

[89] Mozilla. 2026. Navigator: hardwareConcurrency property. https://developer.mozilla.org/en-US/docs/Web/API/Navigator/hardwareConcurrency (accessed: September, 2026).

[90] Mozilla. 2026. Oblivious HTTP (OHTTP) explained. https://support.mozilla.org/en-US/kb/ohttp-explained (accessed: September, 2026).

[91] Mozilla. 2026. Privacy Settings Breakage Study. https://docs.google.com/presentation/d/1OVtXAnyeBLX2N1yyZoTMP9AV_6HnI3mnXwIFlOL7yOA/edit?slide=id.g2c198d8539_0_0#slide=id.g2c198d8539_0_0 (accessed: September, 2026).

[92] Mozilla. 2026. Private Browsing - Use Firefox without saving history. https://support.mozilla.org/en-US/kb/private-browsing-use-firefox-without-history (accessed: September, 2026).

[93] Mozilla. 2026. Resist Fingerprinting. https://support.mozilla.org/en-US/kb/resist-fingerprinting (accessed: September, 2026).

[94] Mozilla. 2026. Screen: availHeight property. https://developer.mozilla.org/en-US/docs/Web/API/Screen/availHeight (accessed: September, 2026).

[95] Mozilla. 2026. StorageManager: estimate() method. https://developer.mozilla.org/en-US/docs/Web/API/StorageManager/estimate (accessed: September, 2026).

[96] Mozilla. 2026. Telemetry Portal. https://telemetry.mozilla.org/ (accessed: September, 2026).

[97] Mozilla. 2026. The Graphics Canvas element. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/canvas (accessed: September, 2026).

[98] Mozilla. 2026. Tor Uplift: Fingerprinting Resistance. https://bugzilla.mozilla.org/show_bug.cgi?id=uplift_tor_fingerprinting (accessed: September, 2026).

[99] Mozilla. 2026. Web Audio API. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (accessed: September, 2026).

[100] Mozilla. 2026. WebGL constants. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Constants#webgl_debug_renderer_info (accessed: September, 2026).

[101] Mozilla Support. 2018. Gmail shows wrong timestamps. https://support.mozilla.org/en-US/questions/1212634 (accessed: September, 2026).

[102] Mozilla Support. 2019. All websites show UTC time. https://support.mozilla.org/en-US/questions/1238935 (accessed: September, 2026).

[103] Mozilla Support. 2021. Uploaded images display with vertical stripes. https://support.mozilla.org/en-US/questions/1322787 (accessed: September, 2026).

[104] Mullvad and The Tor Project. 2026. Mullvad Browser. https://mullvad.net/en/browser (accessed: September, 2026).

[105] Shaoor Munir, Sandra Deepthy Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. 2022. CookieGraph: Understanding and Detecting First-Party Tracking Cookies. In *Computer and Communications Security*. ACM, New York, NY, USA.

[106] Sebastian Neef. 2026. Uncovering Fingerprinting Networks. An Analysis of In-Browser Tracking using a Behavior-based Approach. https://arxiv.org/abs/2210.11300 (accessed: September, 2026).

[107] Hoang Dai Nguyen and Phani Vadrevu. 2025. Breaking the Shield: Analyzing and Attacking Canvas Fingerprinting Defenses in the Wild. In *World Wide Web Conference*. ACM, New York, NY, USA.

[108] Pouneh Nikkhah Bahrami, Umar Iqbal, and Zubair Shafiq. 2021. FP-Radar: Longitudinal Measurement and Early Detection of Browser Fingerprinting. In *Privacy Enhancing Technologies*. De Gruyter, Berlin, Germany.

[109] Nisenoff, Alexandra and Borem, Arthur and Pickering, Madison and Nakanishi, Grant and Thumpasery, Maya and Ur, Blase. 2023. Defining "Broken": User Experiences and Remediation Tactics When Ad-Blocking or Tracking-Protection Tools Break a Website's User Experience. In *USENIX Security Symposium*. USENIX Association, Anaheim, CA, USA.

[110] Kiu Pau, Vicki Lee, Shih Yin Ooi, and Ying Pang. 2023. The Development of a Data Collection and Browser Fingerprinting System. *Sensors* (2023).

[111] Mike Perry, Erinn Clark, Steven Murdoch, and Georg Koppen. 2018. The Design and Implementation of the Tor Browser [DRAFT]. https://2019.www.torproject.org/projects/torbrowser/design/ (accessed: September, 2026).

[112] Privacy Community Group of the W3C. 2026. CHIPS (Cookies Having Independent Partitioned State). https://github.com/privacycg/CHIPS (accessed: September, 2026).

[113] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. 2020. Long-Term Observation on Browser Fingerprinting: Users' Trackability and Perspective. In *Privacy Enhancing Technologies*. De Gruyter, Berlin, Germany.

[114] Takamichi Saito, Takafumi Noda, Ryohei Hosoya, Kazuhisa Tanabe, and Yuta Saito. 2019. On Estimating Platforms of Web User with JavaScript Math Object.

In *Advances in Network-Based Information Systems*. ACM, New York, NY, USA.
[115] Saito, Takamichi and Yasuda, Koki and Kazuhisa, Tanabe and Takahashi, Kazushi. 2018. Web Browser Tampering: Inspecting CPU Features from Side-Channel Information. In *Advances on Broad-Band Wireless Computing, Communication and Applications*. Springer International Publishing.
[116] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. 2024. The Double Edged Sword: Identifying Authentication Pages and their Fingerprinting Behavior. In *World Wide Web Conference*. ACM, New York, NY, USA.
[117] Signifyd. 2026. E-commerce fraud protection platform. https://www.signifyd.com/ (accessed: September, 2026).
[118] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. 2022. The Dangers of Human Touch: Fingerprinting Browser Extensions through User Actions. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.
[119] Junhua Su and Alexandros Kapravelos. 2023. Automatic Discovery of Emerging Browser Fingerprinting Techniques. In *World Wide Web Conference*. ACM, New York, NY, USA.
[120] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. 2009. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.
[121] Paul Syverson and Matthew Traudt. 2018. HSTS Supports Targeted Surveillance. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.
[122] The Tor Project. 2026. The Tor Browser and Network. https://www.torproject.org/ (accessed: September, 2026).
[123] The Tor Project. 2026. Tor at the Heart: Firefox. https://blog.torproject.org/tor-heart-firefox/ (accessed: September, 2026).
[124] Hacker News user. 2020. GoDaddy login fails with resistFingerprinting. https://news.ycombinator.com/item?id=25066631 (accessed: September, 2026).
[125] Reddit user. 2020. YouTube borked with new Firefox. https://old.reddit.com/r/firefox/comments/fy6l1z/youtube_borked_with_new_firefox/ (accessed: September, 2026).
[126] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA.
[127] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *Symposium on Security and Privacy*. IEEE, Piscataway, NJ, USA.
[128] Tanvi Vyas, Andrea Marchesini, and Christoph Kerschbaumer. 2017. Extending the Same Origin Policy with Origin Attributes. In *Conference on Information Systems Security and Privacy*. SciTePress, Setúbal, Portugal.
[129] W3C. 2021. Web Authentication: An API for accessing Public Key Credentials. https://www.w3.org/TR/webauthn/ (accessed: September, 2026).
[130] W3C Privacy Working Group. 2026. Mitigating Browser Fingerprinting in Web Specifications. https://w3c.github.io/fingerprinting-guidance/ (accessed: September, 2026).
[131] Web Hypertext Application Technology Working Group (WHATWG). 2026. HTML Living Standard - dom-localstorage-dev. https://html.spec.whatwg.org/multipage/webstorage.html#dom-localstorage-dev (accessed: September, 2026).
[132] Web Hypertext Application Technology Working Group (WHATWG). 2026. HTML Living Standard - dom-sessionstorage-dev. https://html.spec.whatwg.org/multipage/webstorage.html#dom-sessionstorage-dev (accessed: September, 2026).
[133] WebKit. 2023. Bug 260842: Add general quirk for canvas2d fingerprinting. https://bugs.webkit.org/show_bug.cgi?id=260842 (accessed: September, 2026).
[134] WebKit. 2024. Private Browsing 2.0. https://webkit.org/blog/15697/private-browsing-2-0/ (accessed: September, 2026).
[135] WebKit. 2026. Tracking Prevention in WebKit. https://webkit.org/tracking-prevention/ (accessed: September, 2026).
[136] World Wide Web Consortium (W3C). 2026. Same-Origin Policy (SOP). https://www.w3.org/Security/wiki/Same_Origin_Policy (accessed: September, 2026).
[137] World Wide Web Consortium (W3C). 2026. Service Workers Nightly. https://www.w3.org/TR/service-workers/ (accessed: September, 2026).
[138] 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 *Network and Distributed System Security Symposium*. Internet Society, Reston, VA, USA.
[139] Desheng Zhang, Jianhui Zhang, Youjun Bu, Bo Chen, Chongxin Sun, and Tianyu Wang. 2022. A Survey of Browser Fingerprint Research and Application. *Wireless Communications and Mobile Computing* (2022).

A Ethical Considerations

The benefits of this work — better understanding the risks associated with browser fingerprinting at scale and the benefits from concrete privacy-focused motivations — depend explicitly on obtaining large-scale data from individual users and analyzing that data to understand how different browser features can be used to infer user "uniqueness". However both of these benefits implicate privacy concerns, that we discuss here.

A.1 Data Collection

The telemetry gathered for this study explicitly captures details of individual user's computer and browser configurations and thus, even though nothing about the user's behavior is implicated, the data is still inherently sensitive — it is, in essence, a roadmap dataset for building idealized browser fingerprints. Here we outline the steps taken to address the attendant risks for this collection.

First, we collected this data in accordance with Mozilla's existing data collection policy [87], which emphasizes user privacy, transparency, accountability, and necessity. Users are informed through Mozilla's privacy policy and can opt out of data collection at any time.

For these measurements, we applied additional safeguards to mitigate any potential misuse. First, data collection was limited to a subset of Firefox users (approximately 25 million). Second, all user characteristics were submitted as separate probes via OHTTP [90], a privacy-enhancing protocol that decouples client identity (e.g., IP address) from request content by routing encrypted requests through an intermediary. Third, access to the data was restricted to a small subset of Mozilla employees and all data was retained for at most 90 days. Fourth, users can opt out of telemetry at any time, which also deletes any previously submitted data.

In the context of Mozilla's existing data collection program, we believe that this combination of limiting user exposure, restricting access, and minimizing retention, provides a practical balance between enabling the key benefits of this research while protecting user privacy interests.

A.2 Data Analysis

A second privacy concern results from the product of our analysis. Because our dataset provides a more comprehensive view of browser fingerprinting than is typically available publicly, we deliberately constrained both the analyses we performed and the results we release. These choices are intended to support defenders in evaluating and improving privacy protections, while limiting the risk of enabling new and more effective fingerprinting techniques.

In particular, this is why we choose to focus on modeling existing fingerprinting strategies rather than attempting to derive a strictly more powerful set of fingerprinting features. Similarly, when releasing aggregate results, we restrict disclosures to signals that are already commonly collected by fingerprinting systems.

Finally, fingerprint stability over time is a central factor in practical fingerprinting. While reducing stability can hinder fingerprinting, at its limit this approach converges to randomization (cf. Section 5.2). Although stability is therefore an important dimension, we do not analyze it in this work, as we do not expect such analysis to yield actionable improvements beyond existing defenses.

B Generative AI Usage

Generative AI, specifically Claude Code [9], was used to assist with code development for data collection, analysis of fingerprinting scripts, assessment of web compatibility, and the generation of figures and tables.

As per Mozilla's policy, all data collection code adhered to Mozilla's standards for landing code in Firefox [84], including peer review by a domain expert. Before deployment, initial test data was collected on the Beta channel, which receives several hundred thousand submissions, to verify correctness and data quality.

For analyzing fingerprinting scripts, generative AI supported three complementary techniques. First, scripts were de-obfuscated and statically analyzed to extract observed measurements. Second, scripts were embedded in a controlled HTML environment, where measurement functions were invoked, return values recorded, and resulting payloads captured. Third, using Firefox's JS Tracing feature, all DOM API calls made by the scripts were recorded. Together, these static and dynamic artifacts enabled comprehensive identification of payload fields.

The initial version of the data analysis code (June 2025) was self-authored but limited in efficiency. Generative AI was subsequently used to refactor the code for improved performance on modern hardware. For example, the original single-threaded Python implementation was redesigned to use multiprocessing and free-threaded (GIL-less) Python to better utilize CPU resources. Outputs from both versions were compared to ensure consistency.

Beyond translating existing code to a more efficient implementation, Generative AI was also used to assist in the efficient implementation of certain analyses, including the incremental fingerprint analysis. Generative AI was also used to generate figures and LaTeX tables, with all values verified against source data. AI was not used to generate citations, interpret results, or substantially author the paper beyond minor grammar, wording, and spelling suggestions.

C Open Science

To promote transparency and reproducibility, we make our analysis code and selected aggregate datasets publicly available.[3] Due to the sensitive nature of browser fingerprinting data, raw data cannot be released.

We release privacy-preserving aggregate datasets. These provided artifacts are designed to enable independent validation of our results while supporting future research on privacy protections.

We also share the results of reverse engineering fingerprinting scripts and identifying the characteristics they collect in the wild, as well as the analysis code to allow reviewers and researchers to reproduce our methodology, even where access to underlying data is restricted.

[3] https://anonymous.4open.science/r/fingerprint-mitigation/

D Detailed Experiment Tables

Technique	% Uniq	Uniq Δ	Sim ($\times 10^{-7}$)
Baseline	75.7%	-	24.73
FPT Phase 0 (Pre-2023)			
CPU Count (Experience-Driven)	75.7%	0.0	24.73
WebGL Renderer String	67.6%	8.0	43.13
Patch Version	63.9%	3.7	96.53
FPT Phase 1 (2023-2024)			
Fonts	54.8%	9.2	130.81
Math Functions	54.7%	0.1	131.48
FPT Phase 2 (2025)			
CPU Count (Data-Driven)	51.4%	3.3	179.88
Available Screen Res.	47.9%	3.5	227.09
Font Updates	46.8%	1.1	246.82
FPT Phase 3 (2025-2026)			
WebGL Vendor	45.7%	1.1	283.12
Storage Quota	44.0%	1.6	296.05
FPT Phase 4 (2026 and Beyond)			
Software Rendered Canvas	40.4%	3.7	556.77
Accept-Language	38.2%	2.2	580.63
Timezone	34.9%	3.2	633.32

Table 3: Cumulative Improvement Table (Overall). Population-weighted average across all four operating systems, using *Fingerprint Pro* as the modeled fingerprinter. Uniq Δ is the absolute drop in % Uniq (in percentage points) from the previous row.

Technique	% Uniq	Uniq Δ	Sim ($\times 10^{-7}$)
Baseline	75.7%	-	24.73
FPT Phase 0 (Pre-2023)			
CPU Count (Experience-Driven)	75.7%	0.0	24.73
WebGL Renderer String	67.6%	8.0	43.13
Patch Version	72.1%	3.5	56.95
FPT Phase 1 (2023-2024)			
Fonts	69.3%	6.3	28.03
Math Functions	75.6%	0.0	24.74
FPT Phase 2 (2025)			
CPU Count (Data-Driven)	74.6%	1.1	25.30
Available Screen Res.	73.3%	2.3	32.86
Font Updates	74.9%	0.7	24.94
FPT Phase 3 (2025-2026)			
WebGL Vendor	66.9%	8.8	45.02
Storage Quota	74.9%	0.8	25.33
FPT Phase 4 (2026 and Beyond)			
Software Rendered Canvas	74.5%	1.2	26.35
Accept-Language	74.2%	1.5	26.96
Timezone	74.1%	1.5	26.32

Table 4: Individual Improvement Table (Overall). Population-weighted average across all four operating systems, using *Fingerprint Pro* as the modeled fingerprinter. Uniq Δ is the absolute drop in % Uniq from the Baseline.

	Windows			macOS			Android			Linux		
Technique	**% Uniq**	**Uniq Δ**	**Sim ($\times 10^{-7}$)**	**%**	**Δ**	**Sim**	**%**	**Δ**	**Sim**	**%**	**Δ**	**Sim**
Baseline	82.9%	-	3.16	82.6%	-	11.94	62.0%	-	46.24	86.7%	-	76.05
FPT Phase 0 (Pre-2023 - Experience-Driven and Best Effort)												
CPU Count (Experience-Driven)	82.9%	0.0	3.16	82.6%	0.0	11.94	62.0%	0.0	46.24	86.7%	0.0	76.05
WebGL Renderer String	68.6%	14.3	17.84	79.7%	3.0	14.96	60.6%	1.4	75.30	82.5%	4.2	84.37
Patch Version	67.6%	0.9	18.96	75.5%	4.2	23.83	53.1%	7.5	216.07	79.3%	3.2	106.93
FPT Phase 1 (2023-2024 - Experience-Driven and Best Effort)												
Fonts	49.5%	18.1	88.68	71.4%	4.0	28.72	53.1%	0.0	216.07	79.0%	0.3	107.38
Math Functions	49.3%	0.2	90.06	71.4%	0.0	28.72	53.1%	0.0	216.07	79.0%	0.0	107.38
FPT Phase 2 (2025 - Data-Driven Analysis)												
CPU Count (Data-Driven)	44.1%	5.2	168.26	66.8%	4.7	44.75	52.9%	0.2	238.62	75.1%	3.9	121.50
Available Screen Res.	41.4%	2.7	186.08	41.5%	25.2	498.99	52.9%	0.0	238.62	73.0%	2.1	139.08
Font Updates	39.2%	2.2	226.72	41.5%	0.0	498.99	52.9%	0.0	238.62	73.0%	0.0	139.08
FPT Phase 3 (2025-2026 - Data-Driven Analysis)												
WebGL Vendor	37.2%	2.0	297.26	41.2%	0.3	507.04	52.8%	0.1	238.87	71.2%	1.8	156.67
Storage Quota	35.1%	2.0	318.18	41.1%	0.1	507.71	51.7%	1.1	243.79	68.5%	2.8	169.61
FPT Phase 4 (2026 and Beyond - Data-Driven Analysis)												
Software Rendered Canvas	29.3%	5.8	811.45	39.9%	1.2	545.10	50.1%	1.6	290.20	66.2%	2.3	191.11
Accept-Language	28.3%	1.1	828.16	38.2%	1.7	564.21	46.0%	4.1	329.40	65.5%	0.7	192.86
Timezone	25.7%	2.5	891.26	34.0%	4.2	654.29	42.4%	3.5	363.67	60.0%	5.5	225.07

Table 5: Cumulative Improvement Table. Cumulative impact of fingerprinting protection techniques applied in sequence, using *Fingerprint Pro* as the modeled fingerprinter. % Uniq is the percent of users seen as unique for that operating system. Uniq Δ is the absolute drop in % Uniq (in percentage points) from the previous row to this one. Sim is the Simpson Index ($\times 10^{-7}$) of the cohort distribution after applying this row's protection on top of all prior rows.

	Windows			macOS			Android			Linux		
Technique	**% Uniq**	**Uniq Δ**	**Sim ($\times 10^{-7}$)**	**%**	**Δ**	**Sim**	**%**	**Δ**	**Sim**	**%**	**Δ**	**Sim**
Baseline	82.9%	-	3.16	82.6%	-	11.94	62.0%	-	46.24	86.7%	-	76.05
FPT Phase 0 (Pre-2023 - Experience-Driven and Best Effort)												
CPU Count (Experience-Driven)	82.9%	0.0	3.16	82.6%	0.0	11.94	62.0%	0.0	46.24	86.7%	0.0	76.05
WebGL Renderer String	68.6%	14.3	17.84	79.7%	3.0	14.96	60.6%	1.4	75.30	82.5%	4.2	84.37
Patch Version	82.2%	0.7	3.34	78.7%	3.9	18.16	54.5%	7.5	131.49	84.3%	2.4	90.44
FPT Phase 1 (2023-2024 - Experience-Driven and Best Effort)												
Fonts	70.3%	12.6	9.66	79.7%	2.9	13.62	62.0%	0.0	46.24	86.6%	0.1	76.20
Math Functions	82.8%	0.1	3.19	82.6%	0.0	11.94	62.0%	0.0	46.24	86.7%	0.0	76.05
FPT Phase 2 (2025 - Data-Driven Analysis)												
CPU Count (Data-Driven)	81.0%	1.9	4.05	82.2%	0.4	12.18	62.0%	0.0	46.25	85.4%	1.2	77.57
Available Screen Res.	81.9%	1.0	3.31	61.2%	21.4	106.69	62.0%	0.0	46.24	85.7%	1.0	79.87
Font Updates	81.4%	1.5	3.60	82.6%	0.0	11.94	62.0%	0.0	46.24	86.7%	0.0	76.05
FPT Phase 3 (2025-2026 - Data-Driven Analysis)												
WebGL Vendor	67.2%	15.6	21.23	79.5%	3.1	15.04	60.5%	1.5	75.39	81.3%	5.4	87.21
Storage Quota	82.3%	0.6	3.29	82.6%	0.0	11.95	61.0%	1.1	47.51	85.4%	1.2	77.15
FPT Phase 4 (2026 and Beyond - Data-Driven Analysis)												
Software Rendered Canvas	81.6%	1.3	3.48	82.2%	0.4	12.18	60.6%	1.4	50.21	86.5%	0.2	76.34
Accept-Language	82.5%	0.4	3.21	82.0%	0.6	12.18	58.5%	3.5	52.33	86.5%	0.2	76.16
Timezone	82.1%	0.7	3.30	80.7%	1.9	12.94	59.5%	2.5	49.98	84.8%	1.9	77.44

Table 6: Individual Improvement Table. Impact of each fingerprinting protection technique applied in isolation against the Baseline, using *Fingerprint Pro* as the modeled fingerprinter. % Uniq is the percent of users seen as unique for that operating system. Uniq Δ is the absolute drop in % Uniq (in percentage points) from the Baseline. Sim is the Simpson Index ($\times 10^{-7}$) of the cohort distribution under this protection alone.

E Investigating Causes

Several measurements and protection techniques exhibit disproportionate impact on a single platform. Table 7 pairs each observation with its empirical observation and our conjectured cause.

Observation	Likely Cause
Accept-Language collapses far more uniqueness on Android than on any desktop platform. Phase 4 Accept-Language drops Android uniqueness by 3.5 pp (Table 6) vs. ≤0.6 pp on the other operating systems. The underlying `locale_accept_language` measurement carries ∼100,000 distinct values on Android (68% of which are unique), vs. 5,000–9,000 distinct values on the desktop OSes.	Firefox-Android derives `Accept-Language` from the device's system locale, which carries the user's full region tag (e.g., `pt-BR`, `en-GB`, `es-MX`). Desktop Firefox ships a much smaller fixed set of locale-region combinations, collapsing the long tail of regional variants.
CSS system colors rank #2 in the greedy fingerprint construction on Android (Table 10). Specifically, the `accentcolor` sub-field contains 17,479 distinct values on Android (49% unique), vs. 1,324 on Windows; the `highlight` sub-field contains 17,474 distinct values on Android (49% unique), vs. 1,556 on Windows.	Material You dynamic theming [46] derives accent colors from the user's wallpaper, producing a near-continuous color space exposed via CSS system color keywords.
CSS system colors rank #5 in the greedy fingerprint construction on macOS. Specifically, the `highlight` sub-field contains 13,910 distinct values on macOS (89% unique), vs. 1,556 on Windows.	The macOS Highlight Color setting includes an "Other…" arbitrary-color picker, allowing any RGB value to surface.
CSS system colors rank #4 in the greedy fingerprint construction on Linux. Specifically, three independently customizable sub-fields—`accentcolor`, `highlight`, and `selecteditem`—each contain 14,000–17,000 distinct values on Linux (78–79% unique each), vs. 1,300–1,600 distinct values on Windows.	Linux's theming ecosystem is fragmented across GTK and Qt, multiple desktop environments (GNOME, KDE, XFCE), distributions, and community themes (Dracula, Nord, …), each defining different defaults that users can customize freely.
Available Screen Resolution disproportionately reduces uniqueness on macOS. The Phase 2 mitigation drops cumulative macOS uniqueness by 25.3 pp, vs. only 0–3 pp on the other operating systems.	The macOS Dock's height is set by a continuous icon-size slider (16–128 px) [13], rather than the discrete taskbar sizes on other platforms. On Retina displays, user-selectable display scaling layers additional CSS-pixel granularity on top of each slider position—each scaling mode produces a different rounding of the Dock-adjusted `availHeight`.
The Patch Version mitigation has an outsized effect on Android. Phase 0 Patch Version drops cumulative Android uniqueness by 7.5 pp, vs. ≤4.2 pp on the other operating systems.	Firefox-Android ships on a weekly cadence, while desktop Firefox ships biweekly, so more transient patch versions are simultaneously in flight on Android.
Font fingerprinting is disproportionately effective on Windows and effectively a no-op on Android and Linux. Phase 1 Fonts drops cumulative Windows uniqueness by 18.1 pp, vs. ≤4.0 pp elsewhere.	Third-party Windows software (e.g., Microsoft Office, Adobe Creative Suite) often silently installs additional fonts during setup, producing a diverse per-machine font inventory, and the FPJS probe list specifically targets these Windows-common fonts. None of the 52 FPJS-probed fonts appear in the platform-standard font set on Android or Linux, so all users return an identical empty result there.

Table 7: Conjectured causes for the most disproportionate per-platform observations.

F Fingerprinter Modeling

We performed an analysis of several real-world fingerprinters to determine how they analyze users. We first performed a web crawl that identified fingerprinting scripts based on detection of font fingerprinting or canvas fingerprinting, using heuristics such as number of fonts queried by a web page or detection of known fingerprinting text strings rendered in a canvas.

From this crawl, we identified several candidate fingerprinting scripts seen on multiple different domains. From these candidate scripts, we attempted to identify the company that produced or otherwise distributed the script. We chose *Akamai*, *Signifyd*, and *Fingerprint Pro* based on our ability to identify the company, the prevalence of the script on the web, the popularity of the company, and our ability to test it using techniques similar to [67].

Once chosen, the scripts were downloaded and analyzed to understand what measurements they collected. Scripts were deobfuscated and statically analyzed to extract observed measurements, as well as embedded in a controlled HTML environment, where measurement functions were invoked, return values recorded, and resulting payloads captured. Finally, using Firefox's JS Tracing feature, all DOM API calls made by the scripts were recorded. By cross referencing the results of these three techniques we have high confidence in our ability to accurately simulate the fingerprinters. Table 8 contains a list of the measurements collected by each of them.

Akamai	Fingerprint Pro	Signifyd
Accept-Language	Audio Fingerprint	Audio Compressor Gain
FPJS Canvas Image 1	Canvas WebGL Image	Audio Fingerprint (2)
Available Res.	FPJS Canvas Image 1	Audio Frequency Sum
Timezone Offset	FPJS Canvas Image 2	Audio Time-Domain Sum
CPU Count	CSS System Colors	Audio Unique Samples
User Agent	CSS System Fonts	Canvas Image 1
Max Touch Points	Math Operations	Canvas Image 2
Do Not Track	MathML	Canvas Image 07
FPJS Canvas Image 2	Media Capabilities	Canvas Image 8
Screen Res.	WebGL Params	FPJS Canvas Image 2
Cookies Enabled	WebGL Extensions Hash	Software-Rendered Canvas Image 1
navigator.languages	Allowlisted FPJS Probed Fonts	Software-Rendered Canvas Image 2
Color Depth	FPJS Probed Fonts	Software-Rendered FPJS Canvas Image 2
Platform	Max Touch Points	ClientRects Emoji Font Family
OS/CPU String	OS/CPU String	ClientRects Ghost Dimensions
Pixel Depth	CPU Count	ClientRects Known Dimensions
	Intl.locale	ClientRects Text Font Family
	Timezone Offset	ClientRects Element GBCR
	Timezone Name	ClientRects Element GCR
	Accept-Language	ClientRects Emoji
	Platform	ClientRects Range GBCR
	Cookies Enabled	ClientRects Range GCR
	EME Enabled	CSS System Colors
	PDF Viewer Enabled	CSS System Fonts
	Use Document Colors	Allowlisted Probed Fonts (Variant F)
	Color Scheme	Probed Fonts (Variant F)
	Inverted Colors	Allowlisted Probed Fonts (Variant G)
	Monochrome	Probed Fonts (Variant G)
	Prefers Contrast	Allowlisted Probed Fonts (Variant H)
	Prefers Reduced Motion	Probed Fonts (Variant H)
	Prefers Reduced Transparency	Allowlisted Probed Fonts (Variant I)
	Available Res.	Probed Fonts (Variant I)
	Color Depth	Any-Pointer Type
	Screen Res.	Max Touch Points
	Video Dynamic Range	OS/CPU String
	Available Res. Left	Pointer Type
	Available Res. Top	CPU Count
	Pixel Ratio	Intl.locale
	SDP Codec List	Timezone Offset
	Storage Quota	Timezone Name
	User Agent	Accept-Language
	SpeechSynthesis Voices	Platform
	WebAuthn Capabilities	Cookies Enabled
	WebGL Extensions	Do Not Track
	WebGL Precision Fragment	Global Privacy Control
	WebGL Precision Vertex	Use Document Colors
	WebGL Renderer	Color Scheme
	WebGL Vendor	Inverted Colors
	WebGL Vendor Raw	Monochrome
	WebGPU Fallback Adapter	Prefers Reduced Motion
	WebGPU Max Bind Groups	Available Res.
	WebGPU Max Bind Groups+VBO	Color Depth
	WebGPU Max Bindings/Group	Screen Orientation
	WebGPU Max Buffer Size	Pixel Depth
	WebGPU Max Color Bytes/Sample	Screen Res.
	WebGPU Max Color Attachments	Pixel Ratio
	WebGPU Max Compute Invocations	Storage Quota
	WebGPU Max Compute WG Size X	SVG Bounding Box
	WebGPU Max Compute WG Size Y	SVG Computed Text Length
	WebGPU Max Compute WG Size Z	SVG Emoji Set
	WebGPU Max Compute WG/Dim	SVG Extent Of Char
	WebGPU Max Compute WG Storage	SVG Substring Length
	WebGPU Max Dyn. Storage Bufs	User Agent
	WebGPU Max Dyn. Uniform Bufs	WebGL Renderer
	WebGPU Max Interstage Shader Vars	WebGL Vendor
	WebGPU Max Sampled Textures/Stage	WebGL Vendor Raw
	WebGPU Max Samplers/Stage	WebGPU Fallback Adapter
	WebGPU Max Storage Buf Binding	WebGPU Max Bind Groups
	WebGPU Max Storage Bufs/Stage	WebGPU Max Buffer Size
	WebGPU Max Storage Textures/Stage	WebGPU Max Color Attachments
	WebGPU Max Texture Array Layers	WebGPU Max Compute Invocations
	WebGPU Max Texture Dim 1D	WebGPU Max Compute WG Storage
	WebGPU Max Texture Dim 2D	WebGPU Max Dyn. Storage Bufs
	WebGPU Max Texture Dim 3D	WebGPU Max Dyn. Uniform Bufs
	WebGPU Max Uniform Buf Binding	WebGPU Max Storage Bufs/Stage
	WebGPU Max Uniform Bufs/Stage	WebGPU Max Texture Dim 1D
	WebGPU Max Vertex Attributes	WebGPU Max Texture Dim 2D
	WebGPU Max Vertex Buf Stride	WebGPU Max Texture Dim 3D
	WebGPU Max Vertex Buffers	WebGPU Max Uniform Buf Binding
	WebGPU Min Storage Buf Align	WebGPU Max Vertex Attributes
	WebGPU Min Uniform Buf Align	WebGPU Max Vertex Buffers
	WebGPU Missing Features	WebGPU Missing Features

Table 8: Measurements collected by each modeled fingerprinter (*Akamai, Fingerprint Pro, Signifyd*). Red: measurement is collected by all three fingerprinters. Blue: measurement is collected by *Fingerprint Pro* and *Signifyd* but not *Akamai*.

G Collected Measurements

Characteristics and associated measurements collected. Some measurements are sub-fields expanded from a single JSON-valued measurement; the source measurement name is shown in *italics* in blue beside each such measurement. Measurements highlighted in red are used by at least one modeled fingerprinting service. The **Models** column indicates which services use each measurement: **F** = Fingerprint Pro, **S** = Signifyd, **A** = Akamai.

Characteristic	Models	Measurement
Audio		
Oscillator fingerprint	F	audio_fingerprint
OfflineAudioContext fp.	S	audio_fingerprint2
Unique samples	S	audio_unique_samples
Frequency domain sums	S	audio_float_frequency_sum
	S	audio_float_time_domain_sum
Compressor gain reduction	S	audio_compressor_gain_reduction
Channel count		audio_channels
Frame count		audio_frames
Sample rate		audio_rate
Browser		
User agent string	F, S, A	user_agent
OS/CPU string	F, S, A	hw_oscpu
Platform string	F, S, A	platform
Built-in PDF viewer	F	prefs_pdf_viewer
Build date		build_date
Browser version		version
CSS System Colors		
System color values	F, S	css_system_colors
This field collected individual values for each of the following sub-fields: accentcolor, accentcolortext, activeborder, activecaption, activetext, appworkspace, background, buttonborder, buttonface, buttonhighlight, buttonshadow, buttontext, canvas, canvastext, captiontext, field, fieldtext, graytext, highlight, highlighttext, inactiveborder, inactivecaption, inactivecaptiontext, infobackground, infotext, linktext, mark, marktext, menu, menutext, scrollbar, selecteditem, selecteditemtext, threeddarkshadow, threedface, threedhighlight, threedlightshadow, threedshadow, visitedtext, window, windowframe, windowtext.		
CSS System Fonts		
System font resolved values	F, S	css_system_fonts
This field collected individual values for each of the following sub-fields: arial, caption, courier_new, cursive, fantasy, georgia, helvetica, icon, menu, message_box, monospace, sans_serif, serif, small_caption, status_bar, system_ui, times_new_roman, verdana.		
Canvas		
Rendered bitmap	S	canvas_canvasdata01 and 02
		canvas_canvasdata03 through 06
	S	canvas_canvasdata07 and 08
		canvas_canvasdata09 and 10
	F	canvas_canvasdata11_webgl
	F, A	canvas_canvasdata12_fingerprintjs1
	F, S, A	canvas_canvasdata13_fingerprintjs2
Device pixel ratio		canvas_dpr
Acceleration status		using_accelerated_canvas
		canvas_feature_status
Note: Rendered bitmaps were collected using both hardware and software rendering, doubling the measurements shown above.		
ClientRects (El.)		
getBoundingClientRect	S	clientrects_element_gbcr_01 through 12
getClientRects	S	clientrects_element_gcr_01 through 12
ClientRects (Emoji)		
Emoji glyph positions	S	clientrects_emoji_01 through 06
Emoji font family	S	clientrects_emoji_font_family
ClientRects (Misc)		
Text font family	S	clientrects_text_font_family
Ghost element dimensions	S	clientrects_ghost_dimensions
Known element dimensions	S	clientrects_known_dimensions
ClientRects (Rng.)		
getBoundingClientRect	S	clientrects_range_gbcr_01 through 12
getClientRects	S	clientrects_range_gcr_01 through 12
CPU & Architecture		
CPU architecture		hw_cpu_arch
CPU model details		hw_cpu_model

continued on next page

Characteristic	Models	Measurement
Firefox binary architecture		hw_firefox_binary_arch
Font Detection		
Font probes	F	fonts_fpjs_allowlisted
	F	fonts_fpjs_nonallowlisted
		fonts_variant_a_allowlisted through e
		fonts_variant_a_nonallowlisted through e
	S	fonts_variant_f_allowlisted through i
	S	fonts_variant_f_nonallowlisted through i
Missing fonts list		missing_fonts
Font Preferences		
Configured font names		font_default_default_group font_default_western font_name_monospace_default_group font_name_monospace_western font_name_sans_serif_default_group font_name_sans_serif_western font_name_serif_default_group font_name_serif_western
Configured font sizes		font_size_monospace_default_group, monospace_western, variable_default_group, variable_western, font_minimum_size_default_group, and western
Configured color overrides		font_prefcolor_accentcolor, accentcolortext, canvas, canvastext, highlight, highlighttext, selecteditem, and selecteditemtext
Preference modification flags		font_prefmodified_default, minimum_size, name_list_cursive, name_list_monospace, name_list_sans_serif, name_list_serif, name_monospace, name_sans_serif, name_serif, size_monospace, and size_variable
Document rendering settings	F, S	prefs_use_document_colors font_prefs_browser_display_use_document_fonts font_font_name_list_emoji_modified
Hardware Devices		
Logical processor count	F, S, A	hw_processor_count
Camera count		hw_camera_count
Microphone count		hw_microphone_count
Audio output devices		hw_speaker_count hw_group_count hw_group_count_wo_speakers
Gamepad count		hw_gamepads
FPU control word state		hw_fpu_control_state
Locale & Language		
Intl locale	F, S	intl_intl_locale
System locale		intl_system_locale
Keyboard layout		intl_keyboard_layout
Web-visible timezone	F, S	intl_timezone_web
Web timezone offset	F, S, A	intl_timezone_offset_web
System timezone		intl_timezone
Accept-Language header	F, S, A	locale_accept_language
Navigator languages	A	locale_languages
Machine Model		
Model name		hw_machine_model
Math Operations		
Math calculations	F	math_op_0 through 23 *(math_ops)* math_op_fdlibm_0 through 23 *(math_ops_fdlibm)* math_op_diag_0 through 35 *(math_ops_fdlibm_2)*
MathML		
Element rendered widths	F	mathml_1 through 10 mathml_diag_1 through 19 *(mathml_diag_values)*
MathML font family		mathml_diag_font_family
Modified MathML prefs		mathml_diag_prefs_modified
Media & WebRTC		
ICE candidate order		ice_order
ICE SDP answer		ice_sd
SDP codec list	F	audio_codecs *(sdp_codec_list)*
	F	video_codecs *(sdp_codec_list)*
	F	rtpmap_entries *(sdp_codec_list)*
	F	fmtp_parameters *(sdp_codec_list)*
	F	rtcp_feedback_types *(sdp_codec_list)*
Unsupported codec list	F	media_capabilities_unsupported

continued on next page

Characteristic	Models	Measurement
Non-efficient codec list	F	media_capabilities_not_efficient
Non-smooth codec list	F	media_capabilities_not_smooth
H.264 media support	F	media_h264
Media Queries		
Color scheme preference	F, S	screen_color_scheme
Contrast preference	F	screen_prefers_contrast
Reduced motion preference	F, S	screen_prefers_reduced_motion
Reduced transparency pref.	F	screen_prefers_reduced_transparency
Motion Sensors		
DeviceMotion event		hw_motion_freq and decimals
DeviceOrientation event		hw_orientation_freq and decimals
AbsoluteOrientation event		hw_orientationabs_freq and decimals
Operating System		
OS name		os
OS version		os_version
Linux distribution		os_distro, os_distro_id, and os_distro_version
Privacy Settings		
Cookie behavior	F, S, A	prefs_cookiebehavior
Do Not Track	S, A	prefs_general_donottrack_enabled
Global Privacy Control	S	prefs_general_globalprivacycontrol_enabled
EME / DRM enabled	F	prefs_eme_enabled
Autoscroll		prefs_general_autoscroll
Overlay scrollbars		prefs_general_overlay_scrollbars
Smooth scroll		prefs_general_smoothscroll
Block popups		prefs_block_popups
Zoom settings		prefs_zoom_count
		prefs_zoom_text_only
Changed media prefs		prefs_changed_media_prefs
Screen		
Screen resolution	F, S, A	screen_primary_resolution
	S	screen_primary_orientation
Available screen resolution	F, S, A	screen_primary_avail
	F	screen_web_avail_left
	F	screen_web_avail_top
Color depth	F, S, A	screen_primary_color_depth
	S, A	screen_primary_pixel_depth
Inner resolution	F, S, A	screen_web_inner
	F, S	screen_web_pixel_ratio
Outer resolution	F, A	screen_web_outer
		screen_size_mode
HDR & accessibility	F	screen_primary_video_dynamic_range
	F, S	screen_inverted_colors
	F, S	screen_monochrome
Display info		screen_count
		screen_text_anti_aliasing
		target_frame_rate
Speech Synthesis		
Voice count		voices_count and voices_local_count
Default voice		voices_default
Available voices	F	voices_sha1
		voices_sample
		voices_ssdeep_all, local, and nonlocal
Storage		
Disk quota estimate	F, S	storage_quota
SVG		
Bounding box	S	svg_bbox
Text length	S	svg_computed_text_length
	S	svg_substring_length
Character extent	S	svg_extent_of_char
Emoji rendering set	S	svg_emoji_set
Touch & Pointer		
Primary pointer type	S	hw_pointer_type
Any pointer type	S	hw_any_pointer_type
Max touch points	F, S, A	hw_max_touch_points
Pointer contact area		hw_pointer_width and height

continued on next page

Characteristic	Models	Measurement
Pointer pressure		`hw_pointer_pressure` and `tangential_pressure`
Pointer tilt		`hw_pointer_tiltx`, `tilty`, and `twist`
Touch rotation		`hw_touch_rotation_angle`
Web Authentication		
Authenticator capabilities	F	`hybridTransport` *(webauthn_capabilities)*
	F	`passkeyPlatformAuthenticator` *(webauthn_capabilities)*
	F	`userVerifyingPlatformAuthenticator` *(webauthn_capabilities)*
	F	`relatedOrigins` *(webauthn_capabilities)*
	F	`conditionalCreate` *(webauthn_capabilities)*
WebGL		
Renderer string	F, S	`webgl_renderer`
Renderer string (raw)		`webgl_renderer_raw`
Vendor string	F, S	`webgl_vendor`
	F, S	`webgl_vendor_raw`
Version string		`webgl_version`
Version string (raw)		`webgl_version_raw`
Vertex shader precision	F	`webgl_precision_vertex`
Fragment shader precision	F	`webgl_precision_fragment`
Context parameters	F	`webgl_params`
Extension list	F	`webgl_extensions`
Extension list (raw)		`webgl_extensions_raw`
Extension parameters	F	`webgl_params_extensions`
Context alpha flag		`webgl_alpha`
Context antialias flag		`webgl_antialias`
Context type		`webgl_context_type`
Fragment shader source		`webgl_fragment_shader`
Vertex shader source		`webgl_vertex_shader`
Minimal shader source		`webgl_minimal_source`
Note: Every measurement listed for WebGL was also collected separately for WebGL2, and was recorded for both hardware and software WebGL contexts, quadrupling the measurements shown above. WebGL and WebGL2 context parameters were exploded to 142 individual parameters (71 for hardware and software rendering) for analysis. WebGL and WebGL2 extension parameters were exploded to 20 individual values (10 for hardware and software) for analysis.		
WebGPU		
Adapter type	F, S	`wgpu_is_fallback_adapter`
Missing features bitset	F, S	`wgpu_missing_features`
Texture dimension limits	F, S	`wgpu_maxtexturedimension1d`, `2d`, and `3d`
	F	`wgpu_maxtexturearraylayers`
Bind group limits	F, S	`wgpu_maxbindgroups`
	F	`wgpu_maxbindgroupsplusvertexbuffers`
	F	`wgpu_maxbindingsperbindgroup`
	F, S	`wgpu_maxdynamicuniformbuffersperpipelinelayout`
	F, S	`wgpu_maxdynamicstoragebuffersperpipelinelayout`
Shader resource limits	F	`wgpu_maxsampledtexturespershaderstage`
	F	`wgpu_maxsamplerspershaderstage`
	F, S	`wgpu_maxstoragebufferspershaderstage`
	F	`wgpu_maxstoragetexturespershaderstage`
	F	`wgpu_maxuniformbufferspershaderstage`
Buffer size limits	F, S	`wgpu_maxuniformbufferbindingsize`
	F	`wgpu_maxstoragebufferbindingsize`
	F	`wgpu_minuniformbufferoffsetalignment`
	F	`wgpu_minstoragebufferoffsetalignment`
	F, S	`wgpu_maxbuffersize`
Vertex pipeline limits	F, S	`wgpu_maxvertexbuffers`
	F, S	`wgpu_maxvertexattributes`
	F	`wgpu_maxvertexbufferarraystride`
Render pipeline limits	F	`wgpu_maxinterstageshadervariables`
	F, S	`wgpu_maxcolorattachments`
	F	`wgpu_maxcolorattachmentbytespersample`
Compute pipeline limits	F, S	`wgpu_maxcomputeworkgroupstoragesize`
	F, S	`wgpu_maxcomputeinvocationsperworkgroup`
	F	`wgpu_maxcomputeworkgroupsizex`, `y`, and `z`
	F	`wgpu_maxcomputeworkgroupsperdimension`

H Detailed Incremental Fingerprint Construction

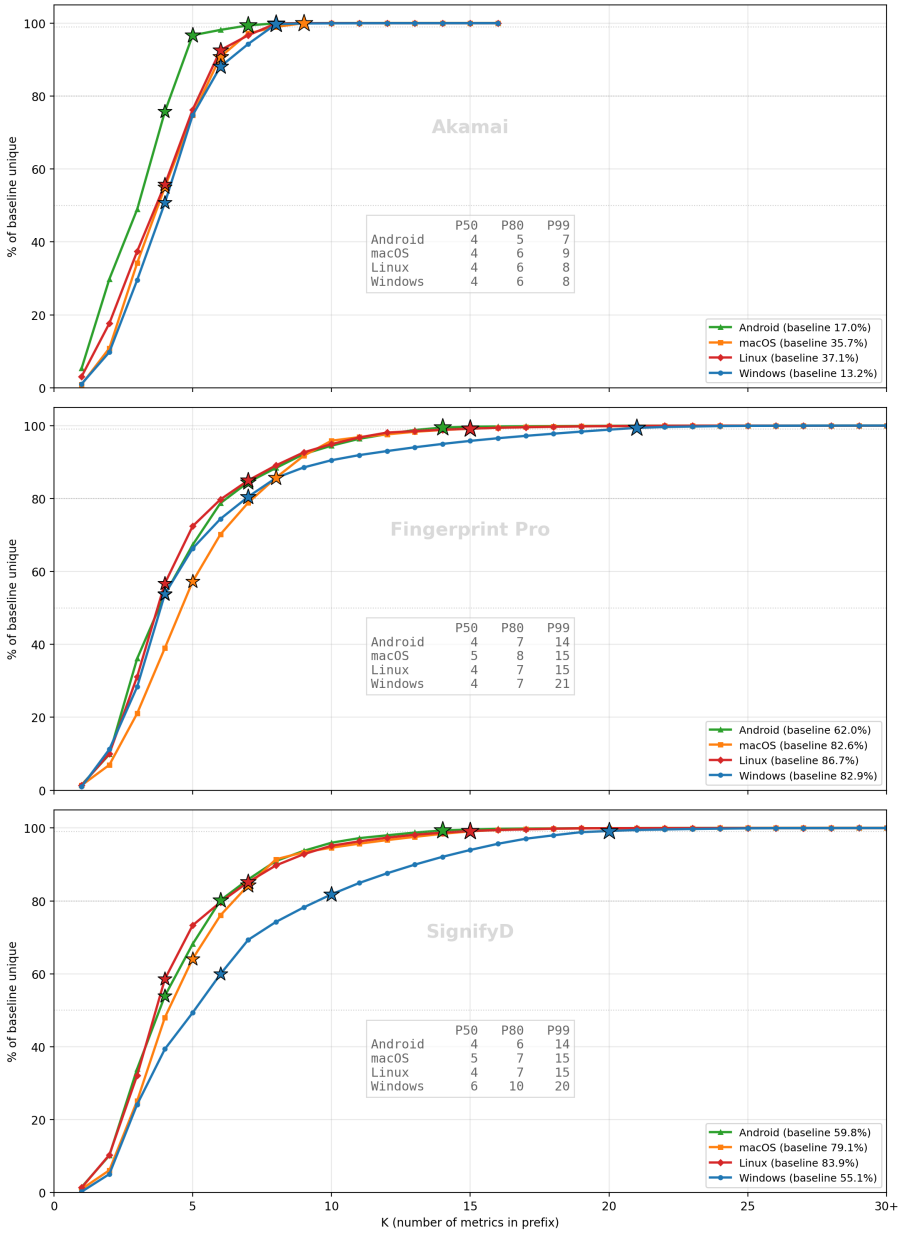


Figure 6: Cumulative coverage of unique users under incremental fingerprint construction, per modeled fingerprinting service (*Akamai*, *Fingerprint Pro*, *Signifyd*) and per operating system. The x-axis is the number of measurements K included so far; the y-axis is the percentage of baseline-unique users captured. Stars mark the P50, P90, and P99 thresholds (also tabulated inset).

	Windows
1	FPJS Probed Fonts
2	WebGL Renderer
3	FPJS Canvas Image 1
4	Timezone Name
5	Available Res.
6	SpeechSynthesis Voices
7	CSS System Colors
8	CPU Count
9	Allowlisted FPJS Probed Fonts
10	Intl.locale
11	Max Touch Points
12	User Agent
13	Storage Quota
14	Audio Fingerprint
15	Prefers Reduced Motion
16	Pixel Ratio
17	Media Capabilities
18	Prefers Reduced Transparency
19	Available Res. Left
20	Color Scheme

	macOS
1	FPJS Probed Fonts
2	Available Res.
3	Timezone Name
4	WebGL Renderer
5	CSS System Colors
6	FPJS Canvas Image 1
7	SpeechSynthesis Voices
8	User Agent
9	Intl.locale
10	Available Res. Top
11	Accept-Language
12	CPU Count
13	Color Scheme
14	MathML
15	Available Res. Left
16	Prefers Reduced Transparency
17	Prefers Reduced Motion
18	Pixel Ratio
19	SDP Codec List
20	Audio Fingerprint

	Android
1	Accept-Language
2	CSS System Colors
3	FPJS Canvas Image 1
4	User Agent
5	Timezone Name
6	WebGL Renderer
7	Screen Res.
8	MathML
9	SpeechSynthesis Voices
10	Pixel Ratio
11	Storage Quota
12	Media Capabilities
13	Prefers Reduced Motion
14	Prefers Reduced Transparency
15	WebGL Params
16	CPU Count
17	Audio Fingerprint
18	Timezone Offset
19	Canvas WebGL Image
20	Color Scheme

	Linux
1	Available Res.
2	WebGL Renderer
3	Timezone Name
4	CSS System Colors
5	MathML
6	FPJS Canvas Image 1
7	User Agent
8	CPU Count
9	EME Enabled
10	Accept-Language
11	Storage Quota
12	SpeechSynthesis Voices
13	Pixel Ratio
14	Color Scheme
15	Available Res. Left
16	Intl.locale
17	FPJS Probed Fonts
18	Prefers Reduced Motion
19	Available Res. Top
20	Media Capabilities

Table 10: Top 20 measurements by iterative incremental fingerprint construction, per operating system, for *Fingerprint Pro*. Rank 1 contributes the most uniquely identifying users on its own; each subsequent rank contributes the most additional uniquely identifying users beyond those already captured by all higher-ranked measurements. Colors indicate shared top-5 measurements: Windows, Android, macOS, Linux.