Understanding Server-side Commercial Fingerprinting

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

Tom Ritter
Mozilla
San Francisco, CA, USA
tom@mozilla.com

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

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

Abstract

Browser fingerprinting is a covert technique for implicitly identifying Web users using combinations of system attributes provided by the browser. However, most studies of fingerprinting have focused on the attributes themselves and how discriminating they *might be*. In this paper, we explore the discriminatory power of fingerprinting in practice, as seen through the lens of a commercial fingerprinting service. Using grey-box testing of the largest commercial fingerprinting service, we selectively mutate inputs to infer key aspects of their approach. In this way, we empirically characterize the relative importance of attributes, and how they are combined with server-side state about cookies and IP addresses to build a fingerprinting service considerably more robust than pure client-side approaches.

CCS Concepts

• **Information systems → World Wide Web**; • **Security and privacy → Privacy protections**.

Keywords

Online Tracking; Browser Fingerprinting

ACM Reference Format:
Elisa Luo, Tom Ritter, Stefan Savage, and Geoffrey M. Voelker. 2026. Understanding Server-side Commercial Fingerprinting. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3774904.3792687

1 Introduction

Web browser fingerprinting is a side-channel technique for distinguishing otherwise anonymous visitors based on a collection of variable attributes (e.g., OS type, fonts installed, etc.). While such technology is sometimes employed to defend against attackers,[1] the more controversial use of fingerprinting is for the purpose of re-identification in support of targeted advertising. It is this use case, and its ability to side-step modern anti-tracking protections, which represents the most significant privacy concern and one of the primary drivers of research in the space.

However, most existing work has focused on this problem from the standpoint of those observed: understanding the relative entropy of different attributes and how uniquely one might be identified via combining them, or measuring the sets of API calls used by fingerprinting scripts in practice. However, modern commercial fingerprinters export extracted browser attributes to stateful services that, together with millions of other fingerprints, and combined with additional data such as cookies and IP address, are used to create concrete "visitor identifiers" for their customers to use. Ultimately, it is the quality of *these* identifiers that represent the current and future privacy impact of fingerprinting in practice.

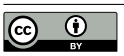
In our work, we focus on fingerprinting through this commercial, server-side lens: how are the underlying features used in practice and how robust is the overall service in the face of anti-fingerprinting functionality? Through empirically analyzing fingerprinting scripts in the wild, we identify commercial services that employ server-side fingerprinting. As a case study, we then become a customer of the Fingerprint Pro service [22], a commercial offering of the most popular vendor in the market. By using Fingerprint Pro, we synthesize a server-side oracle to enable grey-box testing. Then, via carefully controlled experiments, using instrumented browsers to selectively mutate individual attributes, direct management of cookies, and distinct hosting to control for IP address, we infer and empirically map their impact on the service's "visitor identifier".

From this data, our paper offers three contributions: a) identifying the most salient browser attributes in use (i.e., the most determinative on inferred identification), b) how the service integrates the use of exogenous features, in particular cookies and IP address, and c) how robust this overall combination is to existing anti-fingerprinting measures.

2 Background and Related Work

The ability to re-identify individual Web users through the uniqueness of attributes accessible through their browser was first proposed by Eckersley in 2010 [14]. This study was also the first to *quantify* the uniqueness (and thus privacy risk) of individual browser attributes using *information entropy*. There have since been a range of similar studies, offering uniqueness measures between 33% and 90% depending on the number of attributes considered

[1] For example, a bank may use fingerprinting to infer if a visitor to their site is using a *different* machine than they previously used to login [50], and thus invoke additional protections (e.g., against session cookie-stealing attacks). Other sites may use fingerprinting to infer that large numbers of apparently distinct requests likely originate with the *same* machine and thus may reflect (malicious) bot automation.



WWW '26, Dubai, United Arab Emirates

ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792687

Vendor	Type	Attribute Exfiltration
Akamai^S	Server	POST
Fingerprint Pro	Server	POST
mail.ru	Client	–
Imperva^S	Server	POST
Human Security^S	Server	Cookie
Sift Science^S	Server	GET
InsurAds	Client	–
Adscore^S	Server	POST + Beacon

Table 1: Types of fingerprinting services, their method for exfiltrating attributes, and whether they are ^{S}security-oriented.

and how browsers were sampled [3, 29, 35]. Most recently, a 2024 study [6] examined the entropy of *correlated* browser attributes as seen by real-world websites, and confirmed that, in practice, many attributes can be highly correlated and that entropy is a reasonable proxy for fingerprinting risk. All of these studies reveal that the exact data collection and fingerprint computation *methodology* can heavily influence the computed privacy risk of particular fingerprintable attributes.

However, key to the ability to track individual users longitudinally is the stability of fingerprints over time (i.e., the extent changes in software or user environments cause attributes to mutate) [14, 54]. Vastel et al. empirically observed that almost half (45%) of user fingerprints change after a single day, but also provided an algorithm tracking many users for over 50 days [54].

Another line of work has focused on measuring how browser fingerprinting appears in-the-wild. This includes studies measuring the prevalence of browser fingerprinting across the Web, both in the context of user tracking [15] and security uses [13, 50]. Several studies have also assessed the use of particular JavaScript APIs in fingerprinting scripts using static [32] and dynamic [36] methods.

Finally, much of this work is implicitly built around the notion that fingerprints are computed deterministically and statically — i.e., that a client-side hash of browser attributes is sufficient to identify a unique visitor. However, commercial fingerprinters can implement *server-side* fingerprinting. With parallels to server-side tracking [27], server-side fingerprinting scripts *exfiltrate* browser attributes to a *centralized* location and hence can implement a range of re-identification algorithms instead of being limited to exact-hash matching. Thus, the entropy of a particular browser attribute may not reflect its true impact on the ability to identify unique visitors.

However, there has been limited research on the operation of commercial fingerprinting services. To the best of our knowledge, there are only two studies that touch on commercial fingerprinting practices. Nikiforakis et al. [46] analyzed the scripts of three commercial fingerprinting services: BlueCava (now defunct), Iovation, and ThreatMetrix. While they identified several new fingerprinting surfaces (e.g., plugins and Flash), they did not analyze internal algorithms, i.e., *how* these services used such features to perform re-identification. Similarly, Azad et al. [1] performed grey and black-box analyses on commercial *bot detection services* and found that all but one used browser fingerprinting. Again, they focused on the service's efficacy in protecting against common bot-based attacks (e.g., scraping) rather than re-identifying specific users.


```
{ "X0MqRRovL3c=": "df7f17b38f494b11f513ba43327e6549", "JV0QW2M3F2s=":
"c3f34134fb8affb871fdd2b1ace3d9ef", "VQ0gCxBmKzs=": "Mozilla",
"WiZvIBxKaxU=": "Apple M1, or similar", "Ui5nKBdHZhg=": "WebGL 1.0",
"MVEEV3c9BWM=": ["ANGLE_instanced_arrays","EXT_blend_minmax","...",
"WEBGL_provoking_vertex"], ... }
```


Figure 1: Example Human Security fingerprinting payload (truncated, full version in Appendix A.5). Key names are encrypted (decoding the Base64 yields non-sensical results).

This paper focuses on squarely on this understudied aspect: how a server-side commercial fingerprinting service integrates browser attributes and other information in practice.

3 Server-side Fingerprinting

Not all modern fingerprinting services engage in server-side fingerprinting. To identify popular services that do, we used eight of the most popular fingerprinting services identified by Luo et al.'s 2025 study [37], which used canvas rendering behavior to identify specific fingerprinting services.[2] We determined that six of them use server-side fingerprinting (Table 1).

In this section, we describe our method for distinguishing between client and server-side fingerprinting, how server-side fingerprinters exfiltrate browser attribute data, and which browser attributes they exfiltrate. We emphasize that server-side fingerprinting is more privacy-invasive because it transmits user browser attributes to a third party (as opposed to a hash of them).

3.1 Identifying Server-side Fingerprinting

We determined whether a fingerprinting service exfiltrates browser attributes to their servers by capturing the network requests of their scripts executing in browser sessions. For each service, we captured network behavior when visiting multiple customer sites of each service (Appendix A.2). For the sites that we visited, the network behavior was consistent across service customers. We identified six of the eight services as engaging in server-side fingerprinting using a variety of mechanisms for exfiltrating data. The scripts for Akamai, Fingerprint Pro, Imperva, and Adscore perform an HTTP `POST` request to a service endpoint with browser attribute data sent as the request payload. Sift Science performs a `GET` request for a 1x1 pixel gif from `hexagon-analytics.com` with browser attribute data encoded as query parameters (Figure 4 in Appendix A.5). Human Security stores browser attributes in the `px_fp` cookie (and in browser local storage) that is sent to their servers in subsequent requests (that also collect and send behavioral data) [31].

Two of the eight services do not engage in server-side fingerprinting. InsurAds, an advertisement attention and analytics platform, also sets a cookie included in subsequent network requests to their endpoint. However, the cookie only includes an ID computed client-side from the attributes its script collects. Similarly, mail.ru's fingerprinting script computes the hash client-side.

3.2 Browser Attributes Exfiltrated

We analyzed the *browser fingerprinting payloads* of the six server-side fingerprinting services to identify what attributes they exfiltrate. Imperva and Adscore encrypt their payloads, while Akamai,

[2]We included the three most popular services identified in [37] (FingerprintJS, mail.ru, Akamai), which account for the majority of sites (66%) while the remaining have a prevalence of just 1–2% each.

Vendor	# Keys	# Hashed	Features Hashed
Akamai^S	24 → 132	1	canvas (hash of 3 rendered canvas readouts via `toDataURL`)
Fingerprint Pro	41 → 122	8	canvas (geometry + text), math lib, WebGL (parameters, extensions + 3 more)
Human Security^S	65	6	AudioContext (`getChannelData`), 4 canvas readouts, WebGL context parameters
Sift Science^S	72	7	MIME types, plugins, canvas (readout and HTMLCanvas context parameters)

Table 2: Number of browser attributes exfiltrated and how many are hashed. Some keys correspond to multiple browser attributes bundled together in one key-value field (e.g., Navigator properties), and we show the number of bundled attributes expanded from the number of keys as *keys* → *bundled*. ^{S}Denotes a security-oriented service.

Fingerprint Pro, Human Security, and Sift Science have payloads with readable (e.g., Base64 encoded) content.[3] Thus, we focus on the payloads of these four services. This process requires effort because most services send the browser fingerprint payload in JSON format with varying degrees of obfuscation and minification (Figure 1 shows a truncated example of a Human Security payload, and Appendix A.5 shows three complete examples). As a result, the names of keys themselves did not provide much insight into which browser attributes they correspond to (and some attributes are hashed, further obfuscating their identity).

For payload fields with obfuscated keys and unhashed values, we identified the browser attributes associated with the keys by comparing the values in the payloads with the (known) values of the attributes of the browser we used to execute the fingerprinting script. For payload fields that were hashed, we *inferred* which browser attributes they correspond to by individually mutating a wide range of browser attributes (using the framework we will describe in Section 4.2) and looking for consistent correlated changes to one of the payload fields in the fingerprint payload.

Table 2 summarizes the total number of browser attributes contained in the fingerprint payloads, the number of attributes that are hashed, and a short list of hashed attributes (complete list in Appendix A.3). Note that not all fields in the fingerprinting payload correspond to a specific browser attribute (e.g., some are checksums and timestamps).

We highlight hashed attributes because they make it more difficult for fingerprinting services to detect the use of privacy tools (Section 5). Hashed values reduce overhead for the fingerprinter (e.g., raw canvas data can be kilobytes), but they reduce fingerprinter *visibility* into browser behavior. Privacy tools can more freely modify the values of hashed browser attributes (e.g., add random noise) without the fingerprinter detecting either the modifications or the use of that specific tool (which can be a fingerprintable signal itself [11]). Fingerprinters also cannot use hashed values to detect internal inconsistencies within a browser fingerprint [18, 56].

Note that we chose to inspect fingerprinting request payloads directly instead of using browser API introspection (e.g., OpenWPM [15]) because the payloads provide a direct view into the format (e.g., hashed or not) and content of exfiltrated data.

4 Evaluation of Re-identification

In this section we empirically evaluate user re-identification from the perspective of a server-side fingerprinter. We extensively evaluate the behavior of a popular commercial service, and show that server-side fingerprinting can re-identify users even when browser attributes change ("fuzzy matching") — functionality that client-side fingerprinting does not provide. We characterize which browser attributes its re-identification is most sensitive to, and the extent to which server-side re-identification incorporates other more stable features (e.g., IP address, cookies) that are also not typically supported by client-side fingerprinting.

4.1 Fingerprint Pro

For our fine-grained experiments, we became a customer of the *Fingerprint* commercial fingerprinting service. *Fingerprint* is based on the widely-used source-available library FingerprintJS [24], but implements server-side identification techniques [21]. We provide a deep case study of Fingerprint Pro over other commercial fingerprinting vendors for a combination of reasons. It is the only popular service that exposes a computed fingerprint, the "visitor identifier" (VID), to the customer. It is also the only popular service that markets *flexible* use: re-identification of "anonymous users" [16] as well as bot detection and other security use cases. Finally, it has a low barrier for becoming a customer. For the other vendors in Section 3, a live conversation with a salesperson or representative was required to purchase or demo their service.

We purchased the "Pro" tier of Fingerprint's service and integrated their fingerprinting script on a Web site we control. We then performed controlled browser experiments visiting our site, varying browser attributes across visits, and used Fingerprint's server API to programmatically retrieve the result of each "identification event", including the computed visitor identifier.

Additionally, Fingerprint Pro provides a "Browser Tampering" signal: an `anomalyScore` between 0 and 1 reports how improbable a browser fingerprint is based on their statistical model [18].[4] Fingerprint Pro also detects the use of incognito modes, virtualization software, VPNs, privacy-focused settings (e.g., those implemented by the Brave browser), and offers an IP reputation score and bot detection system (based upon the open-source BotD [23] library). These signals are available to customers for every site visit.

Having access to Fingerprint Pro's customer dashboard provides a unique opportunity to perform a grey-box evaluation of a commercial fingerprinting service. Although we do not have visibility into its backend models or algorithms, the visitor ID and browser signals provide a useful form of ground truth. Our findings may not be generalizable to the entire server-side fingerprinting ecosystem (other services may employ different algorithms, etc.), but our case

[3]While Fingerprint Pro also encrypts their payload prior to transmitting to their server, we are able to inspect the payload content server-side (Section 4.1).

[4]Fingerprint Pro by default deems fingerprints with an `anomalyScore` above 0.5 to be engaging in browser tampering.

study of Fingerprint Pro provides an experimental framework for future research in this space.

4.2 Spoofing Browser Fingerprints

Our experiments rely on being able to spoof and mutate arbitrary browser attributes. To implement this functionality, we extended the coverage of `FP-Spoofer` created by Lin et al. [36], a browser extension that can comprehensively spoof particular fingerprinting-related browser properties and APIs.[5]

Specifically, `FP-Spoofer` is able to emulate most Navigator, Window, and Screen properties, and make the HTTP request header consistent with spoofed properties. The extension also can forge canvas fingerprinting-related values `toDataURL` and `getImageData`, WebGL-related values (e.g., `getExtension`, `getParameter`, and `getSupportedExtensions`), emulate font fingerprinting (through spoofing `offsetHeight` and `offsetWidth` properties of span elements), canvas font fingerprinting through CanvasRenderingContext2D's `measureText`, and AudioContext fingerprinting through AudioBuffer's `getChannelData`. In all, `FP-Spoofer` can fool AmIUnique [2] in addition to several commercial, real-world browser fingerprint-based authentication systems [36].

We extended the spoofing capabilities of `FP-Spoofer` to include several new fingerprinting categories that we discovered Fingerprint Pro to be using when inspecting their fingerprint payload (Section 3.2), and that we have not seen reported in previous work. The first spoofs the `getBoundingClientRect` function. Similar to font preference fingerprinting, this technique measures the height and width of rendered MathML (Math Markup-Language) [47] and emojis. The second spoofs WebGL's `getShaderPrecisionFormat` function. Finally, we added the ability to spoof additional WebGL parameter attributes (identified from fields in the fingerprint payload and traced back to `getParameter` calls the script makes).

After setting the desired browser attribute values, we used a browser with the `FP-Spoofer` extension to visit our test Web site. We ensured that `FP-Spoofer` does not trigger any of Fingerprint Pro's bot detection or browser tampering signals by evading common ways a browser may be deemed suspicious (e.g., we evade browser tampering detection via reflection [38] by spoofing the value of `toString` for the native browser APIs we modify).

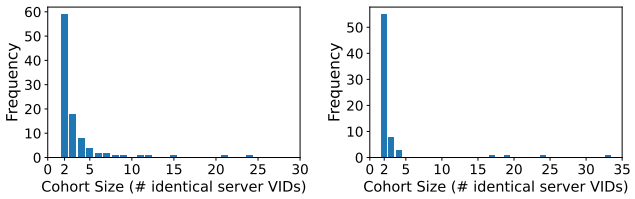
To experiment with many different browser attributes, we used Selenium to automate the process of visiting our site to test the Fingerprint Pro service. We needed to use an anti-detect version of ChromeDriver [53], as Fingerprint Pro is indeed able to detect traditional Web drivers. We confirmed that there was no difference in manually visiting our site versus visiting it with our automated browser by performing several manual checks on our experiments.

4.3 Server vs. Client Fingerprint Uniqueness

To provide an initial framework for understanding what nuances using server-side fingerprinting techniques adds, we analyzed the difference in *fingerprint uniqueness* between Fingerprint Pro and a client-side browser fingerprinting approach (which relies on an *exact* hash of the collected browser attributes).

Methodology. To create thousands of realistic (and internally consistent) browser fingerprints, we used Apify's Fingerprint Suite [4].

[5]Our modifications to `FP-Spoofer` are available upon request.



(a) Mac Cohorts **(b) Windows Cohorts**

Figure 2: Frequency of cohorts of size $n > 1$. Each cohort consists of n browser fingerprints with unique FingerprintJS (client) hashes, but the same Fingerprint Pro (server) VID.

Apify uses a Bayesian network based on distributions of real users' browser attributes to generate realistic browser fingerprints. For each combination of browser attributes generated by Apify, we provided the attributes as inputs to `FP-Spoofer` and visited our test site that uses the Fingerprint Pro service. In total, we made 2,000 visits with unique synthetic fingerprints (1,000 Chrome on MacOS and 1,000 Chrome on Windows).

We ensured that all generated fingerprints are unique under the traditional client-side model of fingerprinting by validating that their open source FingerprintJS hash is unique (by visiting a separate Web page that uses the open-source version of FingerprintJS).

Synthetic fingerprint uniqueness. In total, 66.7% of synthetic MacOS fingerprints had unique Fingerprint Pro visitor identifiers (VIDs), and 76.2% of Windows fingerprints had unique VIDs. For different fingerprint cohorts labeled with the same VID by Fingerprint Pro, Figure 2 shows the distribution of cohort size for cohorts greater than one. The most common cohort size by far is two, meaning that a pair of synthetic browser fingerprints with *different* exact hashes share the same Fingerprint Pro VID.

Least important attributes. For each cohort of size two, we counted the attributes that *were allowed to be different*, yet still resulted in the same VID. Attributes related to screen size (e.g., screen.outerHeight) and browser and OS version (e.g., navigator.fullVersionList) were the most frequent attributes *different* between fingerprints in the same cohort (Table 5 in Appendix A.4).

Most important attributes. Then, for pairs of fingerprints that yielded the same VID, we counted the browser attributes that most commonly *remained the same*.[6] The colorDepth, hardwareConcurrency and pixelDepth attributes remained the same for 100% of cohorts for both MacOS and Windows, and Windows fingerprints had 100% consistency for maxTouchPoints, and MacOS had 100% consistency for deviceMemory (Table 6 in Appendix A.4). Thus, these are attributes that, with high probability, must be the same for Fingerprint Pro to generate the same VID.

Takeaways. We empirically found clear differences in a real-world server-side browser fingerprinting service when compared to a popular client-side library. Further, we provide a first look at the browser attributes that can *differ* between fingerprints with the same server-side VID, e.g., browser window dimensions, which

[6]We removed from consideration any attributes with inherently low diversity, i.e., attributes that would be consistent within all browser fingerprints originating from the same browser-OS combination (e.g., navigator.platform, navigator.vendor), as well as attributes for which Apify generated a very skewed distribution (e.g., screen.hasHDR).

can easily vary even for a given user. The "most important" attributes — those that must be the same within a cohort — correlate to fundamental hardware characteristics that are relatively stable.

4.4 Single-Attribute Modification

To better understand the user re-identification process under a popular instance of the server-side model of browser fingerprinting, we empirically evaluate the effect of modifying individual browser attributes on the Fingerprint Pro visitor identifier.

4.4.1 Methodology. At a high level, we (1) create a unique baseline fingerprint based on a set of browser attribute values; (2) visit our test site that uses Fingerprint Pro to generate a baseline server-side VID_b; (3) modify one specific browser attribute value; (4) revisit the site that uses Fingerprint Pro, generating a VID_m based on the modification, and (5) compare VID_b and VID_m to determine whether Fingerprint Pro successfully re-identified the browser in spite of the attribute modification (VID_b == VID_m) *or* Fingerprint Pro characterized the two sites visits as different users (VID_b != VID_m). We describe each of these steps in more detail.

Generating a Baseline Fingerprint. For each browser attribute we test, we start by creating a baseline fingerprint that we use as a basis of comparison. It is important that the baseline fingerprint is both *unique* (so we are identified as a new user) and *consistent* (so we are not identified as a bot or a user who is tampering with the browser).

To generate unique baseline fingerprints, as in Section 4.3 we used Apify's Fingerprint Suite [4], which generates synthetic (but realistic) browser fingerprints as a set of browser attributes. We then use the server-side information provided by Fingerprint Pro to ensure that the baseline set of browser attributes constitutes a new and consistent fingerprint. If Fingerprint Pro signals the fingerprint as "Browser Tampering", "Bot", or having visited our test site previously, we discard the fingerprint and retry with a different Apify-generated set of attributes. These steps ensure that the baseline fingerprint constitutes a new, unique, and realistic user from the perspective of the server-side fingerprinting service.

Attribute Selection. The browser attributes we started with are known to be used by fingerprinters, including Fingerprint Pro. We include the attributes Laperdrix et al. [34] collected from the fingerprinting literature in 2020. These include HTTP header values (first block in Table 3) and browser information available via JavaScript APIs (block 2). We then added attributes we discovered Fingerprint Pro also collects (Section 3.2), including five `WebGLRenderingContext` attributes (block 3), various APIs never intended to provide discriminatory power between browsers [12] (block 4), and hardware characteristics (block 5).

We note that while most of the attributes that we mutate provide fine-grained discriminatory power between *similar* browsing configurations, a handful (marked in bold in Table 3) fundamentally differentiate between *distinct* browsing configurations. Importantly, modifying the attributes marked in bold in isolation can lead to major internal fingerprint inconsistencies, and hence could be used to detect malicious and/or automated requests [56].

Attribute Modification Test. The goal of the attribute modification experiment is to understand if fine-grained changes to a specific browser attribute results in a new server-side visitor ID.

Attribute	Base	New IP	Private
User-agent (up/downgrade ver)	N	N	N
User-agent (complete change)	T	T	T
HTTP Accept Header	Y	Y	N
HTTP Content Language	B	B	B
Screen Resolution	N	N	N
colorDepth	Y	Y	Y
JS Fonts (List)	N	Y	N
Platform	T	T	T
Do Not Track	N	N	N
Plugins (List)	N	N	N
Canvas (`toDataURL`)	N	Y	N
WebGL Renderer / Vendor	Y	Y	N
WebGL Extensions (List)	Y	Y	N
WebGL Context Attributes	Y	Y	N
WebGL Shader Precisions	N	Y	N
WebGL Context Parameters	N	N	N
Unmasked WebGL Rend./Vendor	Y	Y	N
Math operations	Y	Y	N
AudioContext	N	N	N
Emoji^R	N	N	N
MathML^R	N	N	N
Font preferences^R	N	N	N
hardwareConcurrency	Y	Y	Y
deviceMemory	N	N	N
oscpu^D	T	T	T
vendor	T	T	T

Table 3: Results of modifying a single browser attribute for the baseline ("Base"), different IP address ("New IP"), and VPN ("Private") configurations. Y stands for VID changed; N: VID unchanged (i.e., successful re-identification); T: VID changed + tampering detected; B: VID changed + bot detected. ^{R}getBoundingClientRect-based fingerprinting techniques. ^{D}Now deprecated in Chrome.

After creating and sending the baseline fingerprint, we modified one browser attribute at a time and observed the effect on the returned VID. Specifically, we checked if the VID_m changed relative to the baseline VID_b (indicated by "Y"/"N" in Table 3) and recorded any additional signals Fingerprint Pro's dashboard provides (browser tampering "T" and bot detection "B"). We also validated that the client-side FingerprintJS hash is different after we modified the specific browser attribute (except for "MathML" and "Emoji", which are not used by FPJS). Finally, to validate that the change in VID is not spurious, we reverted to the baseline fingerprint and confirmed Fingerprint Pro returns the same VID as in the baseline step (VID_b). For each browser attribute we mutated, we ensured that the new value was realistic for each attribute (e.g., for deviceMemory we ensured that it was one of the possible values reported [39]). For all except for the bold *fundamental* browser attributes, the exact value we changed the attribute to (so long as it was different) did not matter in terms of the signals visible in the dashboard.

Eliminating Confounding Factors. Each time we changed the browser fingerprint, we used a new browsing session and cleared all cookies between visits. We also performed all experiments from

a variety of campus and residential IP addresses with good reputation (as determined by Fingerprint Pro). Finally, to account for potential differences in the re-identification algorithm between different browsers and OSes, we replicated the experiments with four different Chromium-based configurations: Chrome and Edge on both MacOS and Windows (which have a combined estimated market share of 75% [51]). We found no differences in results among the browser-OS combinations.

4.4.2 Results. Using this methodology, we tested the effect of mutating 26 distinct browser attributes. We describe the results of these tests in terms of each category of attributes.

HTTP headers.[7] The user-agent string provides identifying information about the browser and OS. We tested both changing fine-grained information within the user-agent string (browser or OS version) — which did *not* result in a new VID ("N" in Table 3) — and changing the string completely (a completely different browser or OS) — which resulted in the browser being identified with a different VID ("Y" in Table 3). However, such coarse-grained changes *in isolation* resulted in fingerprint inconsistencies and Fingerprint Pro reported a high browser tampering `anomalyScore` (~0.99). Reporting a browser or OS in the user-agent string that *differs* from values inferred through other browser attributes (e.g., "vendorFlavors" [26]) results in unique VIDs, but Fingerprint Pro understandably deems it suspicious behavior. Modifying the HTTP Accept header (detailing MIME types) in isolation also results in a new VID. While it is not high in information entropy [34] (not highly unique among users), when different it becomes a distinguishing feature. Finally, the HTTP Content-Language header, when modified, results in a new VID but triggers Fingerprint Pro's bot detection (language inconsistencies are a characteristic of automated requests [25]).

Attributes in Fingerprinting Literature. We evaluated eight attributes commonly used in prior fingerprinting studies focused on measurements of entropy. In our experiments, only one (Color Depth) yielded a new VID when changed in isolation. Modifying any of the highest entropy attributes (as reported in [29]), such as the canvas rendering and list of plugins, did not result in new VIDs. One possible explanation is that many anti-fingerprinting tools (that we analyze in Section 5) have focused on these high-entropy attributes, and hence Fingerprint Pro has taken this focus by tools into account in its server-side algorithms.

WebGL Attributes. Many of the `WebGLRenderingContext` attributes we tested resulted in a new VID. While WebGL *rendering* itself is a relatively-studied fingerprinting surface [42], the entropy of attributes accessible through the WebGL *object* (e.g., shaderPrecisions) remains comparatively unknown.

Invasive Fingerprinting Surfaces. Of the "invasive" fingerprinting APIs we tested, only changes in math library operations tested by Fingerprint Pro (e.g., `sin`, `pow`, `PI`) resulted in a new VID.[8]

Hardware Attributes. Changing any of the hardware attributes (except one) resulted in a new VID, but these changes *also* triggered Fingerprint Pro's browser tampering detection. The exception was deviceMemory, which by default browsers already report imprecisely to curtail fingerprinting [39].

Browser Tampering and Bot Detection. None of the attribute changes except for the aforementioned Content-Languages header triggered bot detection. We suspect the reason is that Fingerprint Pro's bot detection is based on static signals like `webdriver` [55]. However, for the *fundamental* browser and hardware attributes (bold in Table 3), modifying them did trigger the browser tampering signal. Hence, Fingerprint Pro does not appear to use fine-grained features to detect browser inconsistencies.[9]

Takeaways. For half (13/26) of the attributes in Table 3, when modified in isolation, client-side fingerprinting returns a new visitor identifier (i.e., the exact hash changes), while a server-side fingerprinter does not. As a server-side fingerprinter, Fingerprint Pro appears to place less weight on browser attributes that have been widely reported as high entropy (e.g., canvas rendering). Critically, while these types of attributes provide high discriminatory power between *very similar* browsing setups, through the lens of a server-side fingerprinter, differences in these features in isolation does not reliably indicate a *different* user. This point is further reinforced by the fact that, when changed *in combination* with IP address, several such attributes yield a new visitor identifier (Section 4.5).

Thus, Fingerprint Pro appears to prioritize re-identification stability over precision. However, it is likely that server-side fingerprinters have enough external information (e.g., cookies, IP address) to still re-identify users with similar fingerprints even *without* many of the high-entropy fingerprinting surfaces. Instead, they can focus on attributes that more certainly indicate a new visitor.

4.5 Exogenous Factors

Unlike client-side fingerprinting, server-side re-identification is not just a straightforward hash of browser attributes. In particular, server-side re-identification can take into account additional factors of the browsing session when deciding on a visitor identifier. Thus, we repeated the above attribute modification tests under three additional browsing conditions: changing the IP address, using private browsing, and allowing cookies and local storage.

IP Address. IP address can be a powerful tracking vector, enabling cross-browser fingerprinting [7] and cookie respawning [28]. Indeed, it can remain relatively stable over time for many users [41], with the caveat that multiple users can share a single IP in various cases (NATs, VPNs, etc.). When used in combination with browser fingerprinting, IP address can enhance user re-identification.

Thus, we tested if modifying the IP address in conjunction with modifying a specific browser attribute changes Fingerprint Pro's server-side re-identification behavior. Using a SOCKS5 proxy managing many IP address ranges, we repeated the attribute modification test while switching to an IP address (after sending the baseline fingerprint) within the same /24, same /16, and same Autonomous System. We also tested with an IP address with a completely different geolocation by using a virtual private network (VPN) (with the caveat that Fingerprint Pro is able to detect the VPN).

As shown in the second results column of Table 3, upon switching to a new IP address outside the same /24, Fingerprint Pro returned

[7]While the browser's `Navigator` interface provides programmatic access to these attributes, Fingerprint Pro reads these values from the HTTP header.

[8]While we used Chrome, we note that Firefox implements standardization of some math operations [44].

[9]For example, while it is possible for canvas rendering to identify particular device types [10], we did not find any evidence of Fingerprint Pro using such information.

new VIDs for three additional browser attributes: Canvas, JS Fonts, and WebGL Shader Precisions. However, there was *no difference* in re-identification behavior from "Base" (keeping the same IP address) when switching between IP addresses within the *same* /24.

Private Browsing. Using a private (incognito) mode when browsing the Web is advertised [30] to be more private. Indeed, the private browsing modes of major browsers offer some heightened privacy measures, including removing all cookies upon beginning a new session. Thus, tracking users across the switch to private browsing mode solely using (first or third-party) cookies becomes impossible. However, browser fingerprinting can be a viable method for tracking users as they switch between normal and private browsing since it does not rely on any locally stored information.

For many of the browser attributes we tested in Section 4.4, modifying them in isolation led to a new VID; however, switching from normal to private browsing *in conjunction* with modifying that browser attribute no longer resulted in a new VID (third results column of Table 3): Fingerprint Pro successfully re-identified the browser switching to private mode. The only two (non-tampering or bot-detection) attributes that still result in a new VID after switching to private browsing are colorDepth and hardwareConcurrency, which correspond to stable hardware traits of the device.

Our tests indicate that Fingerprint Pro has a higher tolerance for *fine-grained* changes to particular browser attributes if they detect the user has switched from standard to private browsing mode. Indeed, some browsers build-in additional fingerprinting protections when using a private mode [5, 43].

Cookies and Local Storage. Unlike browser fingerprints and IP address, which can be ephemeral, a persistent way to track users is to use information stored locally in a user's browser, such as cookies and local storage. For each visit to a site using Fingerprint Pro, its fingerprinting script sets two cookies and two local storage variables. We performed the attribute modification tests again, but instead of starting a new session between sending the baseline and modified fingerprints, we persisted the cookies and local storage variables Fingerprint Pro set for the baseline fingerprint. We also tested keeping the same cookies and deleting the local storage variables, and vice versa.

While the local storage variables had no effect, persisting the cookies Fingerprint Pro sets (`_vid_t` and `_iidt`) overrules *any and all* individual modifications to browser attributes. Even when completely changing the browser fingerprint (new browser, OS, IP address, etc.) but keeping the two cookies, Fingerprint Pro returns the *same* VID (despite it triggering the browser tampering signal).

The `_vid_t` and `_iidt` cookies had a `Samesite` policy of `Lax` and `None`, respectively, so they both can be used in cross-site requests [40]. With these settings, Fingerprint Pro can use the cookies to re-identify users (deterministically) across Web sites. Moreover, both cookies Fingerprint Pro sets have an `Expires/Max-Age` of one year.[10] Given that user fingerprints naturally change and evolve over time [54], the ability to reliably track user fingerprint evolutions over long periods of time via cookies can provide invaluable information for a server-side browser fingerprinter.

[10]In Safari, cookies originating from a 3rd-party IP address have a `Max-Age` of one week [57], but Fingerprint Pro offers ways to circumvent such policies such as by proxying their service through Cloudflare [19].

4.6 Limitations

While we have the advantage of having some ground truth via the information Fingerprint Pro provides in its customer dashboard, we do not have visibility into the model and algorithms they use. In our experiments, we made a best effort to control for several potential confounding factors: we ensured that the baseline fingerprint is always deemed unique, we performed all experiments from campus and residential IP addresses (removing IP reputation as a factor) and repeated experiments for different OS/browser combinations.

However, there may be other factors server-side fingerprinting algorithms use that we were unable to control for. Further, due to the combinatorial complexity, we did not test the effect of modifying multiple attributes, and it is possible that modifying *combinations* of the attributes that did not result in a new VID when modified *in isolation* would result in a new one. There also could be non-determinism in their models that we could not identify and thus control for. Finally, we did not explicitly test the effect of timing between visits: the modified fingerprint was always immediately sent after the baseline, but the time between user visits may also be taken into account by server-side fingerprinting algorithms.

5 Privacy Implications

As a final experiment, we used the attribute modification tests in the context of the most popular anti-fingerprinting defenses (as rated by [34]). Specifically, we empirically evaluated if particular privacy-enhancing tools (PETs)[11] still prevented re-identification through the server-side model of browser fingerprinting (as opposed to a client-side fingerprinter). We visited our test page with a browser configured with a variety of PETs to determine if Fingerprint Pro could still re-identify it between sessions (without using cookies), and Table 4 summarizes our results. Unless otherwise stated, we also performed the visits from the same IP address. As a baseline, we confirmed that all anti-fingerprinting tools successfully defended against the widely-used [37] client-side FingerprintJS [24].

Browser Built-in Defenses. We tested four privacy-oriented browsers, each with different (advertised) levels of privacy protection. First, Firefox's Strict Enhanced Tracking Protection (ETP) mode adds persistent random noise to canvas fingerprints between browsing sessions and offers standardization in the list of fonts reported and math operations [43]. As with the results in Section 4.4, however, Fingerprint Pro is able to fuzzy-match over the randomized canvas (and may have learned to do so because of the very existence of this defense), and it is able to re-identify our browser despite the standardization of fonts and math operations (likely through other attributes and IP address).

Firefox's Resist Fingerprinting (RFP) offers a significantly heightened level of browser fingerprinting defenses, which include the uniform reporting of values returned by *many* fingerprintable browser APIs (e.g., hardwareConcurrency, pixelDepth) and blocking of canvas readback on top of the protections already offered by ETP. These attribute modifications in combination were sufficient to generate new Fingerprint Pro VIDs, preventing re-identification.

[11]We focused on PETs that modified browser attributes, rather than block the fingerprinting script entirely (e.g., ad blockers, which are also commonly evaded by fingerprinters including Fingerprint Pro [20, 37]).

Privacy-Enhancing Tool	Anti-Fingerprinting Measures	Client-Side	Server-Side	Server + VPN
Firefox Strict ETP [43]**	+ Canvas randomization + Font and math operation uniformity	Y	N	Y
FF.resistFingerprinting (RFP) [44]**	+ Many APIs report uniform values	Y	Y	Y
Brave Browser [9]**	+ Some API restriction + randomization	Y	N	Y
Tor browser [52]	+ Heavy API restriction + uniform reporting	Y	Y	Y
Canvas Blocker [33] Canvas Defender [45]	+ Add persistent random noise to Canvas + Or block Canvas API entirely	Y	N	Y
User-Agent Switchers [48, 49]	+ Change browser user-agent string	Y	Y*	Y*
Font [59], AudioContext [58], & WebGL Fingerprint Defender [58]	+ Add random noise to the respective APIs	Y	N	N

Table 4: Efficacy of privacy-enhancing tools against client-side fingerprinting (FingerprintJS), server-side (Fingerprint Pro), and server-side using a VPN. **With Firefox ETP/RFP and Brave, we test re-identifiability between separate browsing sessions. *Only if the UA is switched to a different OS/Browser (which is detected as browser tampering signals due to OS inconsistencies).

Brave is a privacy-oriented browser that defends against fingerprinting by "farbling" (randomizing) the outputs of various browser APIs: HTMLCanvas, AudioContext, plugins, WebGL debug info (e.g., Unmasked Renderer/Vendor) and more [8]. Recall from Section 4.4 that modifying several of the browser attributes that Brave farbles *in isolation* (e.g., Unmasked Renderer, WebGL Shader Precisions) results in a new VID. In our tests with Brave, though, Fingerprint Pro is able to re-identify the browser and returns the same VID across sessions. Indeed, when visiting our test site with Brave, Fingerprint Pro's dashboard signals the use "privacy settings" (only triggered when using Brave and not Firefox's RFP) and most likely adapts their re-identification method with knowledge of Brave's built-in fingerprinting protections [17].

Finally, Tor offers reliable protection against re-identification on the server (including changing the IP address). However, as a trade-off, it has a well-known reputation of breaking site functionality.

Browser Extension-Based tools. We also tested the most popular (by Chrome/Firefox store downloads) anti-fingerprinting browser extensions that specifically mitigate the fingerprinting of *one* browser API. In line with Section 4.4, canvas, font, AudioContext, and WebGL-based defenses are insufficient in isolation to prevent re-identification. For defenses involving the user-agent, switching to a new browser and/or OS combination was sufficient to create a new VID, but resulted in browser tampering signals.

VPNs. Consistent with Section 4.5, the addition of a VPN prevents re-identification by Fingerprint Pro for most of the privacy tools since the user IP address is not stable. The exceptions are the Font, AudioContext, and WebGL Fingerprint Defender tools, where Fingerprint Pro still tolerates a changed IP address.

6 Discussion

This paper was motivated by a desire to understand fingerprinting from the point of a view of a server-side fingerprinting service. As a case study, our empirical measurements of the market-leading Fingerprint Pro revealed a service that is substantially more resilient to changes in underlying browser attributes — even those traditionally reported as high-entropy — than traditional client-side libraries, and substantially more robust to anti-fingerprinting features. Indeed, we showed that, while the client-side implementation of FingerprintJS is effectively blinded by an array of anti-fingerprinting tools, the server-side Fingerprint Pro (which largely uses the same attributes), is able to defeat the majority of them. We find this seemingly minor choice of *where* to implement fingerprint generation — on the client or the server — can have such a significant impact is unintuitive, but indeed reflects a deep and qualitative distinction.

Server-side fingerprinting services can combine raw attribute data with *long-term state* — both about an individual machine as well as the cohort of machines with similar collections of attributes. This, in turn, allows server-side fingerprinters to automatically train their fingerprinting algorithms using cookies (when available) and stable IP addresses as implicit supervisory signals.[12] Moreover, given the known challenges of fingerprint stability [54], the availability of a large state database, with constant feedback, provides precisely the signals needed to adapt to organic changes in browser attributes (e.g., due to browser or OS updates) and allows the selective use of lower-entropy attributes where necessary instead of a "one size fits all" model inherent in the client-side approach. Finally, any imperfections in fingerprint stability or specificity can be corrected in part using IP address data (evident in our measurements) — again, only possible due to large-scale server-side state. Together, these capabilities create substantive advantages for server-side fingerprinting implementations and, thus, it is these settings that are a more accurate reflection of real-world privacy concerns.[13]

7 Acknowledgements

We thank our anonymous reviewers for their insightful suggestions and feedback. Many thanks also to Cindy Moore and Jennifer Folkestad for operational and administrative support of our research. Funding for this work was provided by NSF grant CNS-2152644, the Irwin Mark and Joan Klein Jacobs Chair in Information and Computer Science, the CSE Professorship in Internet Privacy and/or Internet Data Security, a generous gift from Mozilla, and operational support from the UCSD Center for Networked Systems.

[12]The cookies that Fingerprint Pro sets last for one year, providing them with the ability to monitor browser fingerprint changes over time and, thus, track users over time and space (i.e., as they have purview over many customer sites).

[13]This observation also creates a research challenge, as the massive attribute databases at the core of services such as Fingerprint Pro are not available to the scientific community. Thus, reasoning about server-side fingerprinting setting requires further inference efforts (beyond those in this paper) or the development of large-scale open source fingerprint datasets — which bring their own risks and complexities.

References

[1] Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, and Nick Nikiforakis. 2020. Web Runner 2049: Evaluating Third-Party Anti-bot Services. In *Proceedings of the 17th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (Lisbon, Portugal) *(DIMVA'20)*. 135–159.

[2] AmIUnique.org. 2025. Am I Unique? https://amiunique.org.

[3] Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. 2021. A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication. *ACM Trans. Web* 16, 1, Article 4 (Sept. 2021), 62 pages.

[4] Apify. 2025. fingerprint-generator (part of Fingerprint Suite). https://github.com/apify/fingerprint-suite/tree/master/packages/fingerprint-generator.

[5] Apple. 2023. Apple announces powerful new privacy and security features. https://www.apple.com/newsroom/2023/06/apple-announces-powerful-new-privacy-and-security-features/.

[6] Enrico Bacis, Igor Bilogrevic, Robert Busa-Fekete, Asanka Herath, Antonio Sartori, and Umar Syed. 2024. Assessing Web Fingerprinting Risk. In *Proceedings of the ACM Web Conference* (Singapore) *(WWW '24)*. 245–254.

[7] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. 2011. User Tracking on the Web via Cross-Browser Fingerprinting. In *Proceedings of the 16th Nordic Conference on Information Security Technology for Applications* (Tallinn, Estonia) *(NordSec'11)*. 31–46.

[8] Brave Software. 2020. Fingerprinting defenses 2.0. https://brave.com/privacy-updates/4-fingerprinting-defenses-2.0/.

[9] Brave Software. 2024. Sunsetting Strict Fingerprinting Mode. https://brave.com/privacy-updates/28-sunsetting-strict-fingerprinting-mode/.

[10] Elie Bursztein, Artem Malyshev, Tadek Pietraszek, and Kurt Thomas. 2016. Picasso: Lightweight Device Class Fingerprinting for Web Clients. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices* (Vienna, Austria) *(SPSM '16)*. 93–102.

[11] Amit Datta, Jianan Lu, and Michael Carl Tschantz. 2019. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In *Proceedings of the The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. 351–362.

[12] Disconnect. 2024. Tracker Protection Lists — We Track the Trackers. https://disconnect.me/trackerprotection.

[13] Antonin Durey, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2021. FP-Redemption: Studying Browser Fingerprinting Adoption for the Sake of Web Security. In *Proceedings of 18th Conference on the Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'21)*. 237–257.

[14] Peter Eckersley. 2010. How Unique Is Your Web Browser?. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies* (Berlin, Germany) *(PETS'10)*. 1–18.

[15] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS'16)*. 1388–1401.

[16] Fingerprint. 2023. How to Identify Anonymous Website Visitors. https://fingerprint.com/blog/identifying-unique-and-anonymous-visitors/.

[17] Fingerprint. 2025. Audio fingerprinting: What it is + how it works with Web API. https://fingerprint.com/blog/audio-fingerprinting/#brave

[18] Fingerprint. 2025. Browser Tamper Detection — Smart Signals Reference. https://dev.fingerprint.com/docs/smart-signals-reference.

[19] Fingerprint. 2025. Custom Subdomain Setup — Fingerprint Docs. https://dev.fingerprint.com/docs/custom-subdomain-setup.

[20] Fingerprint. 2025. Evading ad blockers with proxy integrations. https://dev.fingerprint.com/docs/protecting-the-javascript-agent-from-adblockers.

[21] Fingerprint. 2025. FingerprintJS vs. Fingerprint Identification. https://github.com/fingerprintjs/fingerprintjs/blob/master/docs/comparison.md.

[22] Fingerprint. 2025. Identify Every Visitor. https://fingerprint.com/.

[23] FingerprintJS. 2025. BotD: Bot detection library that runs in the browser. https://github.com/fingerprintjs/BotD.

[24] FingerprintJS. 2025. Fingerprint JS. https://github.com/fingerprintjs/fingerprintjs.

[25] FingerprintJS. 2025. languages_inconsistency.ts at commit 7e8988243fa33b8583b6728cd4bd37a5f6af677c. https://github.com/fingerprintjs/BotD/blob/7e8988243fa33b8583b6728cd4bd37a5f6af677c/src/detectors/languages_inconsistency.ts.

[26] FingerprintJS. 2025. vendor_flavors.ts — FingerprintJS. https://github.com/fingerprintjs/fingerprintjs/blob/700787c4c0b945ecbd32a8571fd91465ff379e28/src/sources/vendor_flavors.ts.

[27] Imane Fouad, Cristiana Santos, and Pierre Laperdrix. 2024. The Devil is in the Details: Detection, Measurement and Lawfulness of Server-Side Tracking on the Web. In *Proceedings of the 24th Privacy Enhancing Technologies Symposium* (Bristol, UK) *(PETS'24)*. 16 pages.

[28] Imane Fouad, Cristiana Santos, Arnaud Legout, and Nataliia Bielova. 2022. My Cookie is a Phoenix: Detection, Measurement, and Lawfulness of Cookie Respawning with Browser Fingerprinting. In *Proceedings of the 22nd Privacy Enhancing Technologies Symposium* (Sydney, Australia) *(PETS'22)*. 79–98.

[29] 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 *Proceedings of the 2018 ACM Web Conference* (Lyon, France) *(WWW'18)*. 309–318.

[30] Google Support. 2025. Browse in Incognito mode. https://support.google.com/chrome/answer/95464?hl=en&co=GENIE.Platform%3DDesktop.

[31] HUMAN Security. 2025. Use of cookies & web storage. https://docs.humansecurity.com/applications/use-of-cookies-web-storage.

[32] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2021. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *Proceedings of the 2021 IEEE Symposium on Security and Privacy* (San Francisco, CA, USA) *(S&P'21)*. 1143–1161.

[33] joue.quroi. 2025. Canvas Blocker — Fingerprint Protect. https://chromewebstore.google.com/detail/nomnklagbgmgghhjidfhnoelnjfndfpd.

[34] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting: A Survey. *ACM Trans. Web* 14, 2, Article 8 (April 2020), 33 pages.

[35] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P'16)*. 878–894.

[36] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. 2022. Phish in Sheep's Clothing: Exploring the Authentication Pitfalls of Browser Fingerprinting. In *Proceedings of the 31st USENIX Security Symposium (USENIX Security 22)* (Boston, MA). 1651–1668.

[37] Elisa Luo, Tom Ritter, Geoffrey M. Voelker, and Stefan Savage. 2025. Canvassing the Fingerprinters: Characterizing Canvas Fingerprinting Use Across the Web. In *Proceedings of the ACM Internet Measurement Conference* (Madison, WI, USA) *(IMC'25)*. 993–1001.

[38] madtech (AdTechMadness). 2019. JavaScript Tampering — Detection and Stealth. https://adtechmadness.wordpress.com/2019/03/23/javascript-tampering-detection-and-stealth/.

[39] MDN Web Docs. 2025. Navigator.deviceMemory. https://developer.mozilla.org/en-US/docs/Web/API/Navigator/deviceMemory.

[40] MDN Web Docs. 2025. Third-party cookies — Privacy on the Web. https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Third-party_cookies.

[41] Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Martin Lopatka. 2020. Don't Count Me Out: On the Relevance of IP Address in the Tracking Ecosystem. In *Proceedings of The Web Conference* (Taipei, Taiwan) *(WWW '20)*. 808–815.

[42] Keaton Mowery and Hovav Shacham. 2012. Pixel Perfect: Fingerprinting Canvas in HTML5. In *Proceedings of the Workshop on Web 2.0 Security & Privacy* (San Francisco, CA, USA) *(W2SP'12)*.

[43] Mozilla Support. 2025. Firefox's protection against fingerprinting. https://support.mozilla.org/en-US/kb/firefox-protection-against-fingerprinting.

[44] Mozilla Support. 2025. Resist Fingerprinting. https://support.mozilla.org/en-US/kb/resist-fingerprinting.

[45] Multilogin / Canvas Defender. 2024. Canvas Defender (No-Canvas Fingerprinting) — Firefox Add-on. https://addons.mozilla.org/en-US/firefox/addon/no-canvas-fingerprinting/.

[46] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'13)*. 541–555.

[47] PrivacyCheck (LRZ Sec). 2025. Fingerprinting MathML. https://privacycheck.sec.lrz.de/active/fp_ml/fp_ml.html.

[48] random-user-agent.com. 2025. Random User-Agent (Switcher). https://chromewebstore.google.com/detail/random-user-agent-switche/einpaelgookohagofgnnkcfjbkkgepnp.

[49] Ray. 2025. User-Agent Switcher and Manager. https://addons.mozilla.org/en-US/firefox/addon/user-agent-string-switcher/.

[50] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. 2024. The Double Edged Sword: Identifying Authentication Pages and their Fingerprinting Behavior. In *Proceedings of the ACM Web Conference* (Singapore) *(WWW'24)*. 1690–1701.

[51] StatCounter Global Stats. 2025. Browser Market Share Worldwide. https://gs.statcounter.com/browser-market-share.

[52] The Tor Project. 2025. Anti-Fingerprinting — Tor Browser Manual. https://tb-manual.torproject.org/anti-fingerprinting/.

[53] ultrafunkamsterdam. 2025. undetected-chromedriver: Optimized Selenium Chromedriver patch (Zero-Config) that avoids bot detection. https://github.com/ultrafunkamsterdam/undetected-chromedriver.

[54] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *Proceedings of the IEEE Symposium on Security and Privacy* (San Francisco, CA, USA) *(S&P'18)*. 728–741.

[55] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. 2020. FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the*

Web (San Diego, CA, USA) *(MADWeb'20)*. 13 pages.
[56] Hari Venugopalan, Shaoor Munir, Shuaib Ahmed, Tangbaihe Wang, Samuel T. King, and Zubair Shafiq. 2025. FP-Inconsistent: Measurement and Analysis of Fingerprint Inconsistencies in Evasive Bot Traffic. In *Proceedings of the ACM Internet Measurement Conference* (Madison, WI, USA) *(IMC '25)*. 134–149.
[57] WebKit Contributors / whsieh. 2022. "Cap cookie lifetimes to 7 days for responses from third party IP addresses" (Pull Request #5347). https://github.com/WebKit/WebKit/pull/5347.
[58] Yubi. 2025. AudioContext Fingerprint Defender. https://chromewebstore.google.com/detail/audiocontext-fingerprint/pcbjiidheaempljdefbdplebgdgpjcbe.
[59] Yubi. 2025. Font Fingerprint Defender. https://chromewebstore.google.com/detail/font-fingerprint-defender/fhkphphbadjkepgfljndicmgdlndmoke.

A Appendix

A.1 Ethics

This work does not involve human subjects, nor is there any attempt to infer any information about individuals. Our measurements are taken on our own behalf and do not violate the integrity of Fingerprint Pro's service offering. Moreover, our results are in service to the most vulnerable stakeholders, Web users whose privacy is violated in spite of even diligent efforts to protect it.

A.2 Customer Sites

When analyzing whether a fingerprinting service engages in server-side fingerprinting, we analyzed the network behavior of several customer sites of each service (i.e., sites that use the commercial fingerprinting service) as identified by [37]. Below we list the sites we analyzed for each service. For Fingerprint Pro, we confirmed that the network behavior we observed on their customer sites matched what is described in their provided documentation.

Akamai: costco.com, walmart.com, nvidia.com, st.com, rei.com
Fingerprint Pro: vogue.com, onecountry.com, keepthatpump.com, www.bonappetit.com, wired.com
mail.ru: 1cloud.ru, akibank.ru, helix.ru, infowatch.ru, skynet.ru
Imperva: employflorida.com, iowaworks.gov, junemountain.com, kofc.org, trailfinders.com
Human Security: columbiasportswear.it, tvbythenumbers.com
Sift Science: bumbleandbumble.com, iwantmyname.com, tradesy.com
Adscore: 560theanswer.com, gold-cup.news, neural.one, wokespy.com

A.3 Browser Attributes Exfiltrated

Figure 3 lists all fingerprinting-related browser attributes Akamai, Human Security, Sift Science, and Fingerprint Pro exfiltrate to their servers, based on the data contained within their fingerprinting payloads. For each service, we mark in bold the attributes that are hashed. We exclude any attributes that were not explicitly browser fingerprinting related (e.g., behavioral metrics, checksums).

A.4 Synthetic Fingerprint Attribute Importance

Table 5 shows the attributes, within cohorts of size two (i.e., pairs of fingerprints that have the same Fingerprint Pro visitor ID), that were most frequently allowed to differ. The second and third columns show the percentage of cohorts that had the listed attribute be different between the two fingerprints. Note that we de-duplicated highly-correlated browser attributes (e.g., navigator.fullVersionList and the uaFullVersion HTTP header) when ranking the top attributes shown in the table.

Table 6, on the other hand, shows the frequency a particular attribute within a cohort *remained the same* between the two fingerprints. Note that we removed any attributes with inherently

Akamai
Screen: inner, outer, pageOffset, availHeight, availWidth, colorDepth, pixelDepth.
Functionalities: XDomainRequest, createPopup, removeEventListener, globalStorage, openDatabase, indexedDB, attachEvent, ActiveXObject, dispatchEvent, addBehavior, addEventListener, detachEvent, fireEvent, MutationObserver, HTMLMenuItemElement, Int8Array, postMessage, querySelector, getElementsByClassName, images, compatMode, documentMode, all, now, contextMenu
Battery: charging, chargingTime, dischargingTime, Infinity, level, onchargingchange, onchargingtimechange, ondischargingtimechange, onlevelchange
Navigator: userAgent, appName, appCodeName, appVersion, appMinorVersion, product, productSub, vendor, vendorSub, buildID, platform, oscpu, hardwareConcurrency, language, languages, systemLanguage, userLanguage, doNotTrack, msDoNotTrack, cookieEnabled, geolocation, vibrate, maxTouchPoints, webdriver, plugins
Window.chrome: window.chrome, app, isInstalled, InstallState, DISABLED, INSTALLED, NOT_INSTALLED, RunningState, CANNOT_RUN, READY_TO_RUN, RUNNING
Other (plaintext): Local Storage, Session Storage, Adobe Flash, PDF plugins, JavaScript Version, getTimezoneOffset, encoding of permissions
Other (Hashed) 3 canvas readouts via toDataURL (SHA1 hash)

Human Security
WebGLRenderingContext: vendor, renderer, UNMASKED_VENDOR, UNMASKED_RENDERER, SHADING_LANGUAGE_VERSION, VERSION, SupportedExtensions, parameters
Navigator: userAgent, plugins, mimeTypes, language, languages, cpuClass, platform
Screen: width, availWidth, height, availHeight
Window: indexedDB, openDatabase, devicePixelRatio, localStorage
Other: Date.now()
Other (hashed): AudioContext.getChannelData, 4 canvas readouts via toDataURL, list of WebGL context parameters

Sift Science
Screen: height, width, colorDepth
Navigator: userAgent, productSub, vendor, vendorSub, hardwareConcurrency
Capabilities: javaEnabled, sessionStorage, localStorage, indexedDB, openDB, cpuClass, use of adblocker
Consistency checks: tamperedLanguage, tamperedERsolution, tamperedOS, tamperedBrowser
Other: timezoneOffset, dstOffset, character set, language, number of MIME types, number of plugins, maxTouchPoints, webGL extensions, webGL context parameters
Other (hashed): 3 canvas readouts via toDataURL and canvas properties (MD5 hash), list of MIME types, plugin properties

Fingerprint Pro
Screen: colorDepth, colorGamut, contrast, hdr, height, width, resolution
Capabilities: cookiesEnabled, domBlockers (use of ad blockers), forcedColors, indexedDB, invertedColors, localStorage, monochrome, openDatabase, pdfViewerEnabled, reducedMotion, sessionStorage, touchSupport, privateclickmeasurement
Navigator: cpuClass, deviceMemory, hardwareConcurrency, languages, osCpu, platform, vendor
WebGL: renderer, vendor, UNMASKED_VENDOR, UNMASKED_RENDERER, SHADING_LANGUAGE_VERSION, VERSION
Other: Timezone, getBoundingClientRect (font preferences, emoji, mathML), math library, plugins, winding canvas test
Other (hashed): WebGL – context attributes, extension parameters, list of extensions, context parameters, shaderPrecisions, 2 canvas readouts via toDataURL (geometry and text)

Figure 3: Browser attributes exfiltrated by Akamai, Human Security, Sift Science, and Fingerprint Pro's fingerprinting scripts. Hashed attributes are shown in bold.

Attribute	% Different (MacOS)	% Different (Windows)
screen.outerHeight	94.9%	89.1%
screen.availHeight	91.5%	83.6%
navigator.fullVersionList	66.1%	85.4%
navigator.userAgent	54.2%	61.8%
screen.outerWidth	52.5%	40.0%
window.screenX	50.8%	40.0%
screen.availTop	50.8%	32.7%
navigator.appVersion	42.4%	61.8%

Table 5: Least important browser attributes for server-side re-identification.

Attribute	% Same (MacOS)	% Same (Windows)
screen.colorDepth	100%	100%
navigator.hardwareConcurrency	100%	100%
screen.pixelDepth	100%	100%
navigator.deviceMemory	100%	94.5%
navigator.maxTouchPoints	100%*	100%
screen.height	98.3%	89.1%
WebGL renderer	99.6%	85.5%
screen.width	99.6%	89.1%
screen.availWidth	93.2%	87.2%

Table 6: Most important browser attributes for server-side re-identification. *all MacOS fingerprints have a maxTouchPoints value of 0.

low diversity within a given OS-browser combination (e.g., Navigator.platform, Navigator.vendor), which would be consistent within all browser fingerprints originating from the same browser-OS combination. We also removed screen.hasHDR, for which Apify generated a very skewed distribution. Furthermore, for all MacOS fingerprint, the value of maxTouchPoints was 0.

A.5 Example Fingerprinting Payloads

Figure 4 shows an example of the Sift Science fingerprinting payload, which is formatted as query parameters for a GET request. Figures 5, 6, and 7 show example fingerprinting payloads for Akamai, Human Security, and Fingerprint Pro's server-side fingerprinting services, respectively, which are formatted as JSON and transmitted in the body of a POST request. We found that key names in the payload can be encypted (e.g., in the case of Human Security). It was also common for some *values* in the payload to be hashed prior to transmission (likely to help reduce the size of the overall payload when transmitting larger fingerprinting values). For example, canvas readouts are hashed prior to including in the fingerprinting payload for all of the services we analyzed (e.g., the "cv" variable in Akamai and "KnZfcG8aXUA=" variable in Human Security).

```
https://hexagon-analytics.com/images/611934.gif?bk=a11f14f85d&tm=482
&r=665773604&v=117&cs=UTF-8&h=www.netfirms.com&l=en-US
&S=b3cd4c0052a2a303e9dbe9f25bca1eae
&uu=53a51bc7ebf4d9e2c360463ccae522a
&t=Web%20Hosting%2C%20Domain%20Name%20Registration%20-%20Netfirms.com
&u=https%3A%2F%2Fwww.netfirms.com%2F
&ua=Mozilla%2F5.0%20(Macintosh%3B%20Intel%20Mac%20OS%20X%2010.
&15%3B%20rv%3A143.0)%20Gecko%2F20100101%20Firefox%2F143.0&nm=2
&mh=63196a00446a1e285d1992cfe444aa55&np=5
&ph=332b72bdb211e34e6e3c24f88d7c393b&sh=2160&sw=3840&cd=30
&p=MacIntel&to=420&d=60&ce=true&dt=unspecified&tp=0&ol=true&pr=Gecko
&ps=20100101&vd=&hc=8&je=false&ss=true&ls=true&in=true&db=false
&tl=false&tr=false&ts=false&tb=false&ab=false
&cf=0fa4422fc15b4c487c20e4dc995fb4b7&sri=0
&fph=c0f48806e2dc75d6fcbd91de7957bb6f&fsh=2160&fsw=3840&fcd=30
&fp=MacIntel&ftp=0&fhc=8&fss=true&fls=true&fin=true
&fvch=cdfc4f51de661accf9596408fcc8c962&fad=35.74996626004577&fvf=
&fcg=srgb&ffc=false&fm=0&fc=0&frm=false&fhdr=false
&fmf=1b4539be02333d14ce0a07c372e6a7d4&fa=127&fte=false&fts=false
&fce=true&fpdf=true&fl=1f9d440c44318bf9820279f7e146ddf1
&ft=America/Los_Angeles&pf=348&pfe=282&z=z
```

Figure 4: Complete Sift Science GET request example. Browser attributes are query parameters.


```
{ "ap": true, "bt": {"charging": true, "chargingTime": 0,
"dischargingTime": "Infinity", "level": 1, "onchargingchange": null,
"onchargingtimechange": null, "ondischargingtimechange": null,
"onlevelchange": null}, "fonts": null, "fh": null, "timing": {"1":
13, "2": 333, "3": 495, "4": 606, "profile": {"bp": 1, "sr": 0, "dp":
0, "lt": 0, "ps": 0, "cv": 8, "fp": 0, "sp": 0, "br": 0, "ieps": 0,
"av": 0, "z1": 3, "jsv": 0, "nav": 0, "nap": 1, "crc": 0, "z2": 1,
"z3": 0, "z4": 0}, "main": 606, "compute": 13, "send": 606}, "bp":
[2087755996,1953464915,591862434,325835597,1068473606,-1382186647,
-365096851,-1979186206,-108039040,-1906852049], "sr": {"inner":
[1002,1560], "outer": [1864,1647], "screen": [2743,-707],
"pageOffset": [0,0], "avail": [3840,2061], "size": [3840,2160],
"client": [987,1560], "colorDepth": 24, "pixelDepth": 24}, "dp":
{"XDomainRequest": 0, "createPopup": 0, "removeEventListener": 1,
"globalStorage": 0, "openDatabase": 0, "indexedDB": 1, "attachEvent":
0, "ActiveXObject": 0, "dispatchEvent": 1, "addBehavior": 0,
"addEventListener": 1, "detachEvent": 0, "fireEvent": 0,
"MutationObserver": 1, "HTMLMenuItemElement": 0, "Int8Array": 1,
"postMessage": 1, "querySelector": 1, "getElementsByClassName": 1,
"images": 1, "compatMode": "CSS1Compat", "documentMode": 0, "all": 1,
"now": 1, "contextMenu": 0}, "lt": "1759362090135-7", "ps":
[true,true], "cv": "bb9c1d61427c768256965296a9ad5b500142acdd", "fp":
false, "sp": false, "br": "Chrome", "ieps": false, "av": false, "z":
{"a":1471783815,"b":1,"c":1}, "zh": "", "jsv": 1.5, "nav":
{"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
Safari/537.36", "appName": "Netscape", "appCodeName": "Mozilla",
"appVersion": "5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
Safari/537.36", "appMinorVersion": 0, "product": "Gecko",
"productSub": "20030107", "vendor": "Google Inc.", "vendorSub": "",
"buildID": 0, "platform": "MacIntel", "oscpu": 0,
"hardwareConcurrency": 8, "language": "en-US", "languages":
["en-US"], "systemLanguage": 0, "userLanguage": 0, "doNotTrack":
null, "msDoNotTrack": 0, "cookieEnabled": true, "geolocation": 1,
"vibrate": 1, "maxTouchPoints": 0, "webdriver": false, "plugins":
["PDF Viewer","Chrome PDF Viewer","Chromium PDF Viewer","Microsoft
Edge PDF Viewer","WebKit built-in PDF"]},
"crc": {"window.chrome": {"app": {"isInstalled": false, "InstallState":
{"DISABLED":"disabled","INSTALLED":"installed","NOT_INSTALLED":"not_installed"},
"RunningState": {"CANNOT_RUN":"cannot_run","READY_TO_RUN":
"ready_to_run","RUNNING":"running"}}}}, "t":
"f767c7b0c44ec0cb415549744211e1f603c1c93b", "u":
"e8b53f6a0efc5fe0b065d1e0ea1669fc", "nap": 11311144241322244122 }
```


Figure 5: Complete Akamai fingerprinting payload. Most keys are in plaintext, but a few values are hashed.

```
{ "X0MqRRovL3c=": "df7f17b38f494b11f513ba43327e6549", "JV0QW2M3F2s=":
"c3f34134fb8affb871fdd2b1ace3d9ef", "VQ0gCxBmKzs=": "Mozilla",
"WiZvIBxKaxU=": "Apple M1, or similar", "Ui5nKBdHZhg=": "WebGL 1.0",
"MVEEV3c9BWM=":
["ANGLE_instanced_arrays","EXT_blend_minmax","...","WEBGL_provoking_vertex"],
"JDxROmFUVwE=": ["[1, 1]","[1, 64]",8,"yes",...,"127","127"],
"AEA1BkUqMjY=": "Apple", "KnZfcGwbWEc=": "Apple M1, or similar",
"Czd+cU1bf0c=": "WebGL GLSL ES 1.0", "WiZvIB9KbRE=":
"53d75b5cda61e918133a90a59b8a2190", "KnZfcG8aXUA=":
"11b03d038d25a4ad2a9cfe11e1b7d21d", "KnZfcGwWX0U=":
"f2731af47f3354a7953745886a766d45", "cRFEFzR9QSc=":
"3a909b17763864b522b947cf7e30df2c", "JxsSHWF2Fyg=":
"35.7501022554934", "DzN6dUpbeE4=":
"451a844752c9e884ffcaa777b36bbe3b", "M28GKXYFAR8=":
"f6a14091ddee99c99c4cdf9bba4e77ad", "Ix8WGWZwFik=":
"016beb17dd57a6e446b36265284c0c9c", "STk8fwxRPU0=":
["_bcq","_sessionAIBcq","__next_require__","...","_ptf"],
"HwNqBVppbTY=": ["_reactListeningjwtn480a3rp"], "OAhNTn1kTXk=":
"79125fd8dde9fab801a88be527e9c246", "NAxBSnFgQX4=":
"33c1c920dd913a549ad3e5ac3745f147", "b1NaVSkzXG8=": 1,
"dgoDTDBqCXg=": true, "FU1gS1Ama3A=": true, "STk8fwxTPUw=": false,
"JDxROmJWUg4=": false, "LxMaFWl+HC8=": true, "NAxBSnFkSn4=":
"missing", "cg4HSDdjB30=":
["_bcq","_sessionAIBcq","_pxAppId","...","_pxInit"], "b1NaVSkyXGU=":
["_reactListeningjwtn480a3rp"], "HCQpIllJKBk=": ["PDF
Viewer::Portable Document Format::application/pdf pdf::text/pdf pdf",
"..."], "Az92eUVffUg=": "1759274412195", "JV0QW2A2EGE=":
"yr@https://client.px-cloud.net/PXu6b0qd2S/main.min.js:2:20681...",
"U08mSRYkLHM=": true, "Em4naFcDIVw=": 37, "XiJrJBtJahA=":
"fd7149bbfb316699ef918fa7bb7510a8", "fWVIIzsPShg=":
"d41d8cd98f00b204e9800998ecf8427e", "cg4HSDRiAHs=":
"fd7149bbfb316699ef918fa7bb7510a8", "Ix8WGWZzHC0=": 3, "ifv": 3,
"ift": "1759274412195", "Em4naFQPIVk=": 3840, "S3c+MQ0ZNAI=": 2160,
"MDBFNnVYRQU=": 3840, "MVEEV3Q7DmI=": "3840X2160", "Ew9mCVVlbDs=":
30, "SBh9Xg1xf28=": 30, "Z1tSXSIzUmc=": 2061, "BztyfUFadEY=":
"6ca82c5b", "Ix8WGWVzHCg=": "en-US", "a1deUS46WWs=": "MacIntel",
"EXFkN1cfYwE=": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15;
rv:143.0) Gecko/20100101 Firefox/143.0", "WQksDx9mJjw=":
["en-US","en"], "RT0wewBUMU4=": true, "P2MKJXkMABI=": 8,
"NS0Aa3NHC1k=": 420, "In5XeGQRUkI=": false, "cg4HSDRvAXo=": "Timezone
Information Redacted", "JnpTfGAWVU4=": true, "YGAVZiYMFF0=":
"cef339a3", "Q382OQUSMQk=": "14978cc", "eEgNDj0kDTs=":
"d4acbe702b2ce9d7b185cbf0062c8dea", "NS0Aa3BACl8=": null,
"YQFUByRrUDc=": "9f4ad436c590825b763613cc0227fb6e" }
```


Figure 6: Complete Human Security fingerprinting payload. Key names are encrypted (decoding the Base64 yields nonsensical results).


```
"applePay": "value": -1 , "architecture": "value": 127 , "audio":
"value": 124.04344968475198 , "canvas": "value": "Geometry":
"7429b0f34ddd497d54a0e288ffe9c2d2", "Text":
"b009fbf19d12e63525377d80cf9a6325", "Winding": true , "colorDepth":
"value": 30 , "colorGamut": "value": "p3" , "contrast": "value": 0 ,
"cookiesEnabled": "value": true , "cpuClass": , "deviceMemory":
"value": 8 , "domBlockers": , "emoji": "value": "bottom": 28,
"font": "Times", "height": 18.5, "left": 8, "right": 1288, "top":
9.5, "width": 1280, "x": 8, "y": 9.5 , "fontPreferences": "value":
"apple": 147.5625, "default": 147.5625, "min": 9.234375, "mono":
133.0625, "sans": 144.015625, "serif": 147.5625, "system": 146.09375
, "fonts": "value": [ "Arial Unicode MS", "Gill Sans", "Helvetica
Neue", "Menlo" ] , "forcedColors": "value": false ,
"hardwareConcurrency": "value": 8 , "hdr": "value": true ,
"indexedDB": "value": true , "invertedColors": , "languages":
"value": [ [ "en-US" ] ] , "localStorage": "value": true , "math":
"value": "5963cfe25fe61d0bbd7b4920bc602dc8" , "mathML": "value":
"bottom": 29, "font": "Times", "height": 18.5, "left": 8, "right":
303.140625, "top": 10.5, "width": 295.140625, "x": 8, "y": 10.5 ,
"monochrome": "value": 0 , "openDatabase": "value": false ,
"osCpu": , "pdfViewerEnabled": "value": true , "platform": "value":
"MacIntel" , "plugins": "value": [ "description": "Portable
Document Format", "mimeTypes": [ "suffixes": "pdf", "type":
"application/pdf" , "suffixes": "pdf", "type": "text/pdf" ],
"name": "PDF Viewer" , "description": "Portable Document Format",
"mimeTypes": [ "suffixes": "pdf", "type": "application/pdf" ,
"suffixes": "pdf", "type": "text/pdf" ], "name": "Chrome PDF Viewer"
, "description": "Portable Document Format", "mimeTypes": [
"suffixes": "pdf", "type": "application/pdf" , "suffixes": "pdf",
"type": "text/pdf" ], "name": "Chromium PDF Viewer" ,
"description": "Portable Document Format", "mimeTypes": [
"suffixes": "pdf", "type": "application/pdf" , "suffixes": "pdf",
"type": "text/pdf" ], "name": "Microsoft Edge PDF Viewer" ,
"description": "Portable Document Format", "mimeTypes": [
"suffixes": "pdf", "type": "application/pdf" , "suffixes": "pdf",
"type": "text/pdf" ], "name": "WebKit built-in PDF" ] ,
"privateClickMeasurement": , "reducedMotion": "value": false ,
"screenFrame": "value": [ 40, 0, 70, 0 ] , "screenResolution":
"value": [ 982, 1512 ] , "sessionStorage": "value": true ,
"timezone": "value": "Timezone Value Redacted" , "touchSupport":
"value": "maxTouchPoints": 0, "touchEvent": false, "touchStart":
false , "vendor": "value": "Google Inc." , "vendorFlavors":
"value": [ "chrome" ] , "webGlBasics": "value": "renderer": "WebKit
WebGL", "rendererUnmasked": "ANGLE (Apple, ANGLE Metal Renderer:
Apple M1 Pro, Unspecified Version)", "shadingLanguageVersion": "WebGL
GLSL ES 1.0 (OpenGL ES GLSL ES 1.0 Chromium)", "vendor": "WebKit",
"vendorUnmasked": "Google Inc. (Apple)", "version": "WebGL 1.0
(OpenGL ES 2.0 Chromium)" , "webGlExtensions": "value":
"contextAttributes": "6b1ed336830d2bc96442a9d76373252a",
"extensionParameters": "86a8abb36f0cb30b5946dec0c761d042",
"extensions": "57233d7b10f89fcd1ff95e3837ccd72d", "parameters":
"7bd4d913de3e22461894a997d864dcb8", "shaderPrecisions":
"f223dfbcd580cf142da156d93790eb83"
```


Figure 7: Complete Fingerprint Pro fingerprinting payload. Their script encrypts the payload on the client side before transmitting it to their endpoint.