Assessing Web Fingerprinting Risk*

Enrico Bacis Igor Bilogrevic Robert Busa-Fekete
Asanka Herath Antonio Sartori Umar Syed

Google

Abstract

Modern Web APIs allow developers to provide extensively customized experiences for website visitors, but the richness of the device information they provide also make them vulnerable to being abused to construct browser *fingerprints*, device-specific identifiers that enable covert tracking of users even when cookies are disabled.

Previous research has established *entropy*, a measure of information, as the key metric for quantifying fingerprinting risk. However, earlier studies had two major limitations. First, their entropy estimates were based on either a single website or a very small sample of devices. Second, they did not adequately consider correlations among different Web APIs, potentially grossly overestimating their fingerprinting risk.

We provide the first study of browser fingerprinting which addresses the limitations of prior work. Our study is based on actual visited pages and Web APIs reported by tens of millions of real Chrome browsers in-the-wild. We accounted for the dependencies and correlations among Web APIs, which is crucial for obtaining more realistic entropy estimates. We also developed a novel experimental design that accurately and efficiently estimates entropy while never observing too much information from any single user. Our results provide an understanding of the distribution of entropy for different website categories, confirm the utility of entropy as a fingerprinting proxy, and offer a method for evaluating browser enhancements which are intended to mitigate fingerprinting.

1   Introduction

Modern websites make extensive use of Web APIs to provide sophisticated functionality to their users. Some of these APIs reveal information about the user's device that developers can use to tailor the browsing experience for individual users. For example, a document editing website might only render the fonts that are installed on a user's system, or a video conferencing website might leverage
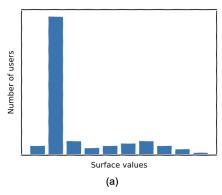
---

*A version of this report to appear in the proceedings of The Web Conference (WWW) 2024. This version contains additional material in the appendix.

a device's graphics card to accelerate performance. However, these granular details can also be used to generate browser *fingerprints*, which are (near) unique identifiers linked to a specific device. Such fingerprints can be used to track users across many different websites, often without their knowledge, and thus pose a major privacy risk. Unlike cookies, fingerprints cannot be easily disabled or reset by users. Although browser fingerprinting can be used to enhance users' online safety (such as preventing fraud or abuse), it is generally perceived as intrusive and therefore discouraged by several popular browsers (e.g., Safari, Firefox, Edge and Brave).

To better assess the risk of browser fingerprinting for users and the effectiveness of anti-fingerprinting methods of browsers, we conducted a large-scale measurement study of Web APIs that focused on quantifying their *entropy*, a measure of information. Each Web API exposes users to a degree of fingerprinting risk that depends on the distribution of its values in the user population. A Web API whose value distribution is very concentrated on a few possible values (Figure 1(a)) provides less identifying information about the average user than a function whose value distribution is very spread out (Figure 1(b)), since sharing values with many users is better for preserving anonymity. Because entropy increases with the spread of a distribution, it serves as an indicator of how well-suited a Web API is for re-identifying users, and has been widely used by researchers as a key metric for quantifying fingerprinting risk (Eckersley, 2010; Laperdrix et al., 2020).

Previous entropy measurement studies suffered from two major limitations. They were either limited to a small sample of websites or devices, and/or they ignored potential correlations among Web APIs, which made it challenging for them to provide realistic estimates of fingerprinting risk. Our novel experimental design allowed us to collect data about dozens of Web APIs called by tens of millions of Chrome browser instances on hundreds of thousands of websites, while limiting the information collected from any single user, thus protecting user privacy. We estimated the correlation between many pairs of Web APIs, enabling us to not only estimate the fingerprinting risk posed by individual Web APIs, but also the joint risk posed by sets of them. Due to the privacy protection mechanisms put in place by our study, we were able to observe telemetry from real user devices instead of an automated crawler, reflecting the real-world behavior of websites as opposed to artificial responses to a bot.

Our results confirmed that Web API usage patterns vary significantly across website categories, and that the information revealed by many Web APIs is often highly correlated. We leveraged these findings to produce the broadest and most realistic measurement of the entropy of Web APIs to date. We validated the use of entropy as a proxy for fingerprinting by showing that it is strongly correlated with known fingerprinting behavior. We also studied how methods that are designed to mitigate fingerprinting can impact the web by quantifying how much information they prevent from being exposed to websites.
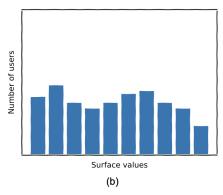
Figure 1: Two possible value distributions for a Web API, (a) low entropy and (b) high entropy.

2   Related Work

Browser fingerprinting has attracted a vast amount of research over the past 15 years (Laperdrix et al., 2020; Zhang et al., 2022). From a broad perspective, prior works that are most related to our study can be grouped in two categories, depending on their contributions:

1. Large-scale measurement studies on browser and attribute uniqueness (Eckersley, 2010; Laperdrix et al., 2016; Gómez-Boix et al., 2018; Andriamilanto et al., 2021a).

2. Attribute set selection studies and inter-attribute correlation (Flood and Karlsson; Fifield and Egelman, 2015; Andriamilanto et al., 2020; Tanabe et al., 2019; Andriamilanto et al., 2021b; Pugliese et al., 2020).

When studying browser uniqueness, it is important to have access to a large and diverse dataset in order to have as many samples from the tail-end of the distribution as possible. These infrequent and uncommon values usually carry the highest amount of identifiable information, and they can have a significant impact on the study results. Therefore, we only discuss prior works that are based on large-scale experiments with real browsers visiting public websites.

Furthermore, as this work focuses exclusively on uniqueness and fingerprinting, we do not discuss the broader topic of online tracking, which is often associated with fingerprinting. A survey on that topic can be found in (Bujlow et al., 2017).

2.1   Browser uniqueness

Considering the vast amount of different configurations of hardware and software components of Internet-connected devices, it is unsurprising to find some configurations to be (nearly) unique. That was the underlying premise of the initial studies of browser uniqueness as observed through web APIs (Laperdrix

et al., 2020). To date, there are four large-scale studies that attempted to measure browser uniqueness in-the-wild (Eckersley, 2010; Laperdrix et al., 2016; Gómez-Boix et al., 2018; Andriamilanto et al., 2021a).

In his pioneering work, Eckersley (2010) created a test site[1] where visitors could choose to have their browser analysed and fingerprinted via HTTP headers, JavaScript and plugins. The two-week experiment gathered over 470k fingerprints, 83.6% of which were unique. This study, which was the first to use the expression "browser fingerprinting", was also the first to use information entropy (in terms of "surprisal") as a proxy for privacy risk on the web, and also among the first to quantify the evolution of fingerprints over time (see Li and Cao (2020) for more recent results on fingerprints evolution over time). They estimated that a browser (in 2010) carried at least 18.1 bits of entropy, with the top-3 properties contributing with (individually) 15.4 bits (plugins), 13.9 bits (fonts), and 10 bits (user agent).

Laperdrix et al. (2016) conducted an analysis of over 118k fingerprints over a 1-year period, by deploying a test website,[2] similar to Eckersley's approach from 6 years before. While, on the one hand, the results confirmed the prior findings as 89.4% of fingerprints were unique, on the other hand they also found significant differences in the prevalence and type of fingerprinting techniques. In particular, the authors found that the availability of the HTML5 canvas element and the wide browser support for webGL APIs contributed significantly to the amount of uniqueness, especially on mobile devices, in the absence of plugins and Flash.

Contrary to the previous two studies (Eckersley, 2010; Laperdrix et al., 2016), which created a dedicated test website where visitors could choose to have their fingerprints collected, Gómez-Boix et al. (2018) collected more than 2 million fingerprints on one of the top-15 most visited French websites. In contrast to the prior results, their results indicated that only 33.6% of all fingerprints were unique, highlighting the potentially large impact that the data collection methodology can have on the findings. Andriamilanto et al. (2021a) conducted a 6-month study where they collected more than 4 million fingerprints from almost 2 million visitors to the same French website as Gómez-Boix et al. (2018). However, this time they collected 216 attributes (as opposed to 17) from a significantly larger user population. In line with prior results (Eckersley, 2010; Laperdrix et al., 2016), the authors observed that 81% of all fingerprints are unique, and more than 90% of the attributes remained unchanged even after 6 months.

While the four studies discussed so far have provided important insights into browser uniqueness, the strength of their results was limited; they only studied uniqueness broadly (i.e., uniqueness of the full fingerprints) or uniqueness of individual attributes (e.g., user agent strings). Yet, different device properties (both hardware and software) are often correlated with each other (Fifield and Egelman, 2015; Andriamilanto et al., 2020; Pugliese et al., 2020), which makes

---

[1]The initial website was `https://panopticlick.eff.org`, which in 2017 moved under `https://coveryourtracks.eff.org`.
[2]`https://amiunique.org`

it crucial to consider them not in *isolation* but *jointly*, as discussed hereafter.

2.2   Attribute set selection

Fifield and Egelman (2015) devised a greedy attribute set selection heuristic that considered the conditional entropy when adding subsequent attributes to a set, in order to obtain the smallest cardinality with the highest fingerprinting potential. Although it provided interesting results, the study was limited in two ways. First, it was based on a very small sample of 1,016 participants from Mechanical Turk. Second, the algorithm only worked on fonts attributes.

Andriamilanto et al. (2020) framed the attribute set selection problem as an optimization problem with two joint objectives: distinctiveness and usability cost. Their greedy algorithm explores a given number of potential sets, by adding attributes one by one to each such set iteratively (similarly to Fifield and Egelman (2015)), and retaining the sets that satisfy the constraints. Although this study used the same large dataset as Gómez-Boix et al. (2018), it still only considered sets that were constructed by adding individual attributes one by one.

Pugliese et al. (2020) conducted a study with 1,304 users and collected 305 attributes, resulting in 88,088 measurements. Their greedy attribute set selection method was based on maximising the stability of trackable attributes as well as the number of participants with trackable attributes, resulting in different sets for mobile and desktop devices. However, the algorithm added attributed one-by-one from the small dataset, and recomputed the objective function until no further improvement can be achieved.

In contrast to related work, our study has a much larger dataset, collected from significantly more websites, a more accurate joint entropy estimation algorithm and better privacy protection. In particular, our study includes data from more than 10 million different Chrome clients who visited more than 100,000 websites in-the-wild. This represents 10x more participants as compared to the other largest study to date (Andriamilanto et al., 2021a), which was done on a single website.

3   Notation and terminology

The web consists of a set of websites $\mathbb{W}$ and the web population consists of a set of clients $\mathbb{C}$, each of which is a browser installation on a single device. A *surface* is a Web API combined with a particular set of input arguments. For example, the Javascript function `WebGLRenderingContext.getParameter(pname)` returns a parameter value of a client's graphics environment, where `pname` specifies the parameter. Let $\mathbb{S}$ be the set of all possible surfaces, and let $x_{s,c}$ be the return value of surface s for client c. Website w observes surface value $x_{s,c}$ if and only if client c visits website w and the site executes a script that invokes the Web API with input arguments corresponding to surface s.

Let client C be chosen uniformly at random from $\mathbb{C}$. For any surface $s \in \mathbb{S}$ define the random variable $X_s = x_{s,C}$, and for any surface set $S = \{s_1, \ldots, s_k\} \subseteq \mathbb{S}$ define the random variable $\mathbf{X}_S = (X_{s_1}, \ldots, X_{s_k})$. In other words, the distribution of $\mathbf{X}_S$ is the distribution of joint values for surfaces S in the client population.

A website can extract different pieces of identifying information from a device over multiple subsequent visits, and therefore it is important to account for this accumulation of information when estimating fingerprinting risk. We define a *session* to be the aggregation of all visits to a specific website by a specific client over a 4 week period.

4 Efficient entropy estimation

The entropy of a discrete random variable X is defined

$$H(X) = -\sum_x \Pr[X = x] \log \Pr[X = x].$$

For any surface set $S \subseteq \mathbb{S}$, the entropy $H(X_S)$ quantifies the fingerprinting risk to users who reveal their values for surfaces in S to websites. The primary objective of our measurement study is to estimate $H(X_S)$ for various surface sets S, a goal that presents at least two major challenges. First, the number of possible surfaces sets is $2^{|\mathbb{S}|}$, and so it is not practical to form an independent estimate for each surface set. Second, for a given surface set S, the number of samples from X_S needed to estimate $H(X_S)$ can be prohibitive. For example, if $|S| = k$, then even if each surface in S has only two possible values, the optimal entropy estimation algorithm needs $\Omega(\frac{2^k}{k})$ samples to estimate $H(X_S)$ accurately (Wu and Yang, 2016).

Instead of directly estimating each $H(X_S)$, we estimate an upper bound based on a well-known decomposition due to Chow and Liu (1968). Let T be any tree whose nodes are the surfaces S. Chow and Liu showed that

$$H(X_S) \leq \sum_{s \in S} H(X_s) - \sum_{(s,s') \in T} I(X_s; X_{s'}) \tag{1}$$

where the second sum is over the edges of T and the mutual information $I(X; Y)$ between random variables X and Y is defined

$$I(X; Y) = \sum_{x,y} \Pr[X = x, Y = y] \log \frac{\Pr[X = x, Y = y]}{\Pr[X = x] \Pr[Y = y]}.$$

Mutual information, which is always non-negative, can be viewed as a measure of the correlation between two random variables. If X and Y are highly correlated then $I(X; Y)$ is large, and if they are independent then $I(X; Y)$ is zero. Thus when surfaces are correlated the Chow-Liu decomposition gives a much tighter upper bound on entropy than the more naive bound $H(X_S) \leq$

$\sum_{s \in S} H(X_s)$, which has been used in previous work (Mendes, 2011). For example, our measurement study found that the Javascript Web APIs that return a user's language setting and keyboard layout have high mutual information (since keyboard layouts are often designed for a particular language), so their joint entropy is significantly lower than the sum of their individual entropies.

At most $|\mathbb{S}|^2$ entropy and mutual information terms must be estimated in order to compute the Chow-Liu upper bound in Eq. (1) for all subsets of $\mathbb{S}$, which is much less than $2^{|\mathbb{S}|}$. As we explain in Section 5, we make the problem even more tractable by not estimating the mutual information between every pair of surfaces, but instead judiciously choosing surface pairs that we expect to be correlated. Finally, each term in the Chow-Liu upper bound can be estimated efficiently. If each surface has $O(v)$ possible values, then the entropy terms need $O(v)$ independent samples to estimate accurately, and the mutual information terms need $O(v^2)$ samples (see Appendix A for a more precise statement of the sample requirements).

5 Experimental methodology

We constructed the set of all surfaces $\mathbb{S}$ by considering all Web APIs exposed via either JavaScript or CSS, manually selecting surfaces that we believed could be sources of stable entropy, while excluding APIs which are gated behind a permission prompt. This led to $\mathbb{S}$ containing a total of 5383 surfaces derived from 161 Web APIs (see Appendix F for complete list of the Web APIs). Note that this approach placed some potential sources of entropy outside the scope of our experiment, such as any information exposed by installed Chrome extensions.

Our experimental goal was to estimate the entropy $H(X_S)$ of many different surface subsets $S \subseteq \mathbb{S}$ subject to the following constraints:

- **Privacy:** Our experiment did not collect too much information from any single user.

- **Performance:** Our experiment did not interfere with the proper rendering of websites or unduly slow the browser.

Given these constraints, we made several key choices when designing our experimental methodology.

- We approximated the entropy $H(X_S)$ of a surface set $S \subseteq \mathbb{S}$ by using the Chow-Liu decomposition which, as we discussed in Section 4, reduces overall sample requirements by an exponential factor.

- We took a 'passive' approach to data collection, by only recording those surface values that were actually observed by websites during client sessions, rather than a more 'active' approach that would execute Web APIs on the client's behalf even if they were not called by any script. Passive sampling prevented performance degradation or unexpected browser behavior due to Web API side effects, and ensured that a user's experience of a website was exactly as the web developer intended.

- We kept the client-side state and logic associated with our experiment as minimal as possible. Each client $c \in \mathbb{C}$ stored only a list of candidate surfaces $L_c \subseteq \mathbb{S}$, a reporting probability $p_c \in [0, 1]$, and the number of reported surface values k_c. Whenever client c visited a website that observed a value for a surface in L_c, with probability p_c that surface value was reported to our servers and the value of k_c was incremented. Surface values were briefly cached on the client before reporting, but were not stored for an extended duration.

- We preserved user privacy by hashing surface values on the client before reporting them, discarded any surface values reported by fewer than 50 clients, and deleted all unaggregated data after 4 weeks. Note that hashing had no impact on entropy estimation accuracy, since entropy is invariant to renaming the surface values. We also never collected more than 20 bits of device-specific information from any client (we explain below how this requirement was enforced). Only a small fraction of Chrome clients were included in our experiment, accounting for less than 10 million clients. Clients had to have the following two settings enabled in Chrome to be included in the experiment: (1) 'Share usage reports and crash analytics with Google', and (2) 'Make searches and browsing better / Sends URLs of pages you visit to Google' (Chrome, 2022).

We conducted the experiment in several phases, with the results of each phase informing the design of subsequent phases:

- **Phase 1:** In order to obtain a high-level understanding of Web API usage, we set $L_c = \mathbb{S}$ and $p_c = 1$ for each client c, but only reported which surfaces were called on the client, and not the surface values (not even hashed versions of the values). No device-specific information was collected during this phase.

- **Phase 2:** We partitioned the surfaces $\mathbb{S}$ into 10 large "families" (see Section 6.1) based on which surfaces we expected to be correlated with each other. We set L_c for each client c to one of the 10 families, thereby increasing the likelihood of observing joint samples from pairs of surfaces from the same family, which in turn led to tighter estimates of the Chow-Liu upper bound (Eq. (1)). We also set the reporting probability $p_c = \min\{1, \frac{40}{n}\}$, where n is the expected number of surfaces in client c's assigned family that will be called during the experiment, a value determined by the data collected in Phase 1. However, once k_c (the number of surface values reported by client c) reached 40, the value of p_c was set to 0. The effect of choosing these values for p_c is that the typical client reported an unbiased random sample of their surface values, but no client reported more than 40 surface values. Note that setting a significantly higher initial value for p_c would have skewed the reports towards surfaces that are called earlier in the experiment (because of the 40 report limit per client) and biased the data collection. We conservatively choose a 40 report limit during

this phase based on an earlier, much smaller and unpublished experiment which estimated that the average surface exposes 0.5 bits of entropy.

- **Phase 3:** While the previous phase collected enough data to estimate the entropy of each individual surface, we found that it was insufficient to estimate the mutual information between many pairs of surfaces (particularly surfaces from two different families), essentially because the probability of jointly observing a pair of surface values from the same client c is only p_c^2. So we augmented our data collection with an additional sampling phase that focused on popular Web APIs. We identified a set of 60 popular surfaces (the 'core', see Section 6.2) and created many overlapping subsets of these surfaces, with each subset constructed so that its entropy was expected to be low, based on the data collected in Phase 2. For each client c we let L_c be one of these subsets and let $p_c = 1$. We estimated that no client reported more than 20 bits of information in Phase 3. Note that the clients used in this phase were disjoint from the clients used in the previous phase.

We used the individual surface entropy estimates and pairwise mutual information estimates computed from the data collected in Phases 2 and 3 to calculate an upper bound on the entropy $H(X_S)$ for many surface sets $S \subseteq \mathbb{S}$, based the Chow-Liu decomposition. When forming the decomposition, we omitted mutual information terms with insufficient samples, as determined by the derivations in Appendix A (note that omitting mutual information terms does not impact the validity of the upper bound in Eq. (1)). Our final estimates of an upper bound for each $H(X_S)$ were accurate to within 1.5 bits with 90% confidence.

Some aspects of our experimental methodology have been omitted from the description in this section for the sake clarity. Complete details can be found in Appendices B, C, D and E.

6 Results

The findings of our measurement study can be grouped into several levels of analysis. First, we calculated how often different families of surfaces (fonts, media, etc) are called by websites, which provided a high-level understanding of how Web API functionality is used across the web. Next, we estimated the entropy of individual surfaces, and the correlation and mutual information between pairs of surfaces, and clustered together surfaces that expose similar identifying information. Finally, we estimated the joint entropy of all the surfaces observed during each client session on each website. This allowed us to plot the entropy distribution for all sessions, validate entropy as a fingerprinting metric, and evaluate the effectiveness of anti-fingerprinting methods based on blocking scripts.
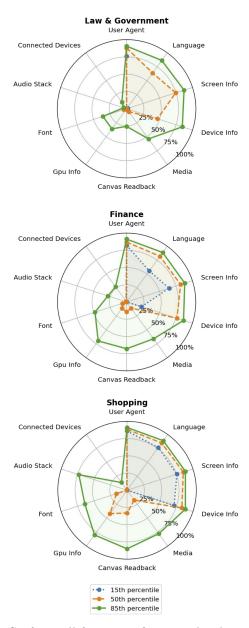
9

Figure 2: Surface call frequency for several website verticals

6.1 Surface call frequency

We partitioned all surfaces into 10 families, and mapped each website to a vertical using a procedure similar to the one used for the Topics API (Dutton, 2023). We defined, for each surface family f, a threshold t_f, and we calculated

the percentage of sessions in which at least t_f different surfaces from surface family f was accessed on a website (recall the definition of a session from Section 3). In the radar charts in Figure 2 we plot several percentiles of these percentages for a few website verticals. In each of these charts, the color indicates the percentage of websites, while the radial dimension indicates the percentage of sessions. For example, Figure 2(c) shows that 15% of shopping websites access screen information in 75% of sessions.

Observe that website behavior can significantly vary across verticals. For example, GPU information is much more commonly accessed by shopping websites than law & government websites, while nearly all websites access information about the user agent.

6.2   Surface clustering

Generally speaking, users are exposed to fingerprinting risk whenever a website calls many high entropy surfaces during a browsing session. But this risk is reduced if the surfaces are correlated with each other, since in that case their joint entropy will be less than the sum of their individual entropies (see Eq. (1)). We calculated all pairwise correlations between 60 'core' surfaces that were selected for their popularity, stability and non-triviality. Specifically, a surface was deemed to be a core surface if over a 1 month period: (1) it was called by at least 500 clients, (2) at least 95% of clients reported only one value for the surface, (3) the surface's entropy was at least 0.1 bits, and (4) the number of clients reporting the surface did not vary significantly each day. The last requirement was intended to filter out surfaces that are only ephemerally popular, such as a canvas drawing function that is temporarily invoked with an unusual set of arguments by a popular website (recall from Section 3 that a surface is a Web API combined with its input arguments).

Defining the correlation between two discrete-valued random variables, such as a pair of surfaces, is not as straightforward as for real-valued random variables, for which metrics like Pearson correlation are typically used. Our approach is based on the observation that two random variables are independent if and only if their joint distribution is equal to the product of their marginal distributions. We applied a method from Chan et al. (2014) to estimate the distance between these distributions, as measured by total variation. Figure 3 reports the results for all pairs of core surfaces, with black squares indicating correlation (total variation is at least 0.05), red squares indicating independence (total variation is zero), and an empty square if there is not enough data to distinguish between these two cases with confidence at least 90%. The entropy of each surface is shown on the right-hand side of Figure 3. An advantage of our definition of correlation is that the sample requirements for estimating it are fairly modest.

We applied the Cuthill–McKee algorithm (Cuthill and McKee, 1969) to reorder the surfaces so that correlated surfaces tend to be adjacent to one another in the figure, and after grouping adjacent and related surfaces together, the following surface clustering emerged:
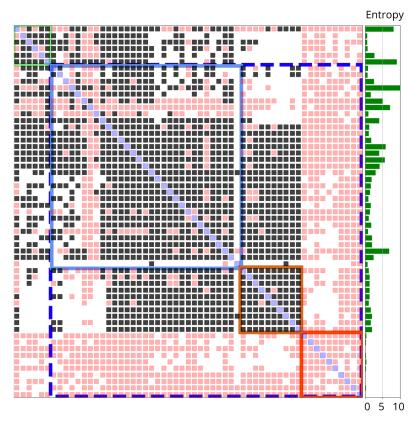
Figure 3: Surface clustering determined by pairwise correlations. Clusters are outlined by colored rectangles.

- **Device properties** *(dashed blue rectangle)*. This cluster includes basic parameters of the device, such as the screen width and height, and is divided into three sub-clusters:

    - **Core device properties** *(light blue rectangle)*. Includes language, pixel ratio, and screen width and height.

    - **WebGL parameters** *(brown rectangle)*. Includes the WebGL parameters.

    - **Independent properties** *(red rectangle)*. Although these surfaces are not correlated with any other surface, they have very low entropy, and so they pose minimal fingerprinting risk.

- **User agent cluster** *(green rectangle)*. Surfaces which return information about the user agent. This cluster also has some cross-cluster correlation with device properties.
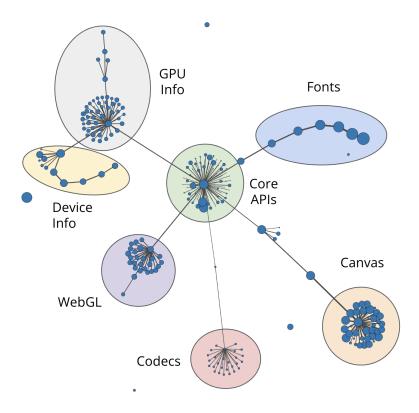
Figure 4: Surface clustering determined by mutual information. Clusters are indicated by ovals.

Expanding this analysis to all surfaces also reveals a natural clustering. Figure 4 is a single-linkage clustering (Everitt et al., 2011) of all the surfaces. In the figure, each node represents a surface, a node's size is proportional to the surface's entropy, and the weight of the edge between two nodes is the mutual information between the surfaces. Observing many surfaces from the same cluster will, to a first approximation, expose a user to less fingerprinting risk than observing the same number of surfaces from different clusters.

6.3 Entropy of sessions

Computing the correlation structure of surfaces also enabled us to estimate the joint entropy of all surfaces observed during any given session (recall the definition of a session from Section 3). Specifically, if S is the set of surfaces observed during a session, then we estimate $H(X_S)$ using the Chow-Liu decomposition in Eq. (1). Since these estimates account for the mutual information between pairs of surfaces, they are more realistic than the estimates provided by any
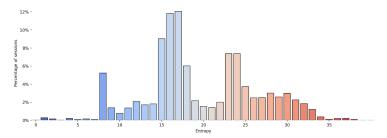
Figure 5: Entropy distribution of the web

previous measurement study. Figure 5 shows the distribution of session entropy for the Tranco Top 10,000 websites (Pochat et al., 2018). Sessions from the same website contribute to multiple buckets of this distribution, but the total contribution is normalized to be the same as all other websites. Note that the entropy for nearly all websites is either far below or far above 20 bits, suggesting that the typical website can be classified as operating in either a high- or low-entropy mode.[3]

Figure 6 decomposes the session entropy distribution into several distributions according to each website's vertical. The distributions are normalized in the same way as the overall distribution in Figure 5, but together they cover only 7187 websites instead of 10,000, since not all websites are assigned to a vertical, and we dropped verticals with fewer than 50 websites. Examining these distributions reveals clear differences between many of the verticals. For example, news and shopping websites have significantly higher entropy consumption than law & government websites.

We can obtain additional insight into differences among verticals by examining whether entropy is exposed in a first- or third-party context, which depends on the domain of the document that observes the surface values. A document in an iframe is first-party if it is same-site (i.e. shares the same registrable domain) with the top level URL (i.e. the URL which the user is visiting), and is third-party otherwise.[4] Figure 7 plots the average first- and third-party entropy per vertical. A typical reason for embedding third-party iframes is to run advertising-related code, and so the differences between the entropy consumption patterns of verticals may be attributable to differences in business models. For example, news websites have the highest third-party entropy, possibly because of their heavy reliance on third-party advertising, while shopping websites have very low third-party entropy, and also seldom display third-party ads. Similarly, users of online communities (which include social networks) are

---

[3]For context, note that if surface values are uniformly distributed in a population of 1 billion users, then (roughly) 10 bits of entropy corresponds to every user sharing their values with 1 million users, 20 bits to 1 thousand users, and 30 bits to every user being uniquely identified.

[4]Note that, even in a first-party context, the information collected via high-entropy APIs can be sent to a third-party (and even directly collected by a third-party script loaded in the first-party context), so this distinction is an approximation.
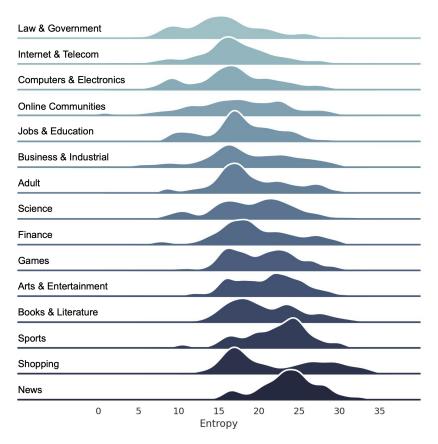
Figure 6: Per-vertical distributions of session entropy, sorted in increasing order of mean of the distribution.

often logged in, which decreases the need to fingerprint them, and this may explain why online community websites consume less overall entropy than news websites.

6.4   Entropy as a fingerprinting metric

While entropy is widely accepted as a proxy for fingerprinting risk, there has been limited data-driven evaluation of its suitability for that purpose. Acar et al. (2014), based on earlier work by Mowery and Shacham (2012), found that many fingerprinting libraries write specific and unusual strings to the canvas, because the way this string is rendered on a user's device can be very identifying. Since these Web API function call 'signatures' constitute clear evidence of known fingerprinting behavior, we checked whether they are associated with higher entropy. Figure 8 is a box plot of the percentage of sessions with a fingerprinting signature versus their average session entropy. We observe a strongly positive
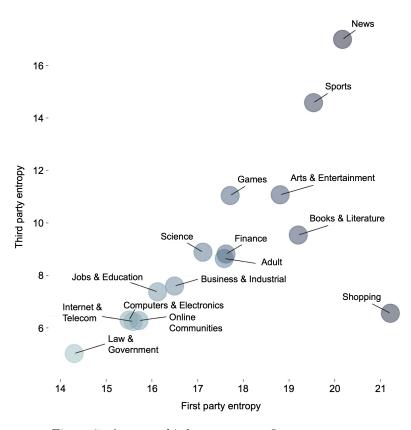
Figure 7: Average third-party versus first-party entropy

correlation between the presence of signatures and entropy.

6.5 Evaluating anti-fingerprinting methods

One approach to mitigating fingerprinting is for the browser to block the execution of scripts that are suspected of fingerprinting. Any such mitigation strategy must navigate a trade-off between limiting the amount of identifying information revealed to websites on the one hand, and disrupting users' experience of websites on the other hand. The results of our entropy measurement study can help quantify this trade-off. We instrumented a crawler to visit 6.4 million websites and record the arguments passed to every function called by every executed script, as well as the domain that hosts each script included in the page. We also calculated the entropy of the information that each website requested from the crawler during its visit.

We next simulated the impact of two fingerprinting mitigation strategies on this entropy distribution. Figure 9 shows the effect of blocking scripts that are hosted by a domain on the Disconnect list of invasive fingerprinters, which is
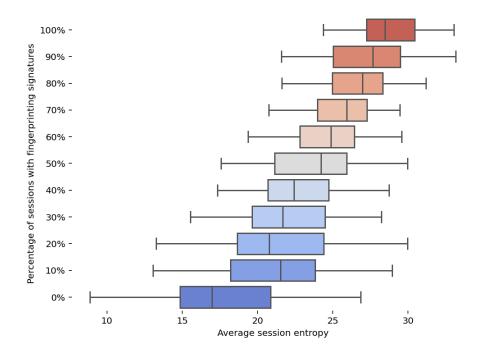
Figure 8: Fingerprinting signatures versus entropy

used by some browsers as the basis of an anti-fingerprinting solution, and has been manually curated to minimize impact on valid website behavior (Disconnect, 2020). Figure 10 shows the effect of the much more aggressive intervention of blocking every script containing at least one of the fingerprinting signatures found by Acar et al. (2014) (see Section 6.4 for further discussion of these signatures), irrespective of its impact on user experience. Note that Figures 9 and 10 show the change in the entropy distribution relative to no intervention. Also, unlike in previous plots, the figures are not based on actual client sessions, but instead on crawler data, since for privacy reasons we did not record the detailed script execution history of any client belonging to a real user. We see that both interventions shift the entropy distribution to lower entropy, but the effect of the more aggressive intervention is much more significant. Browser developers can use this information to design their preferred intervention.

7 Caveats

Our results should be understood in light of the several limitations that had to be imposed on our measurement study, some for practical reasons, and others more fundamental. For the sake of both privacy and performance, we intentionally restricted the scope of our data collection. We examined only a subset of all
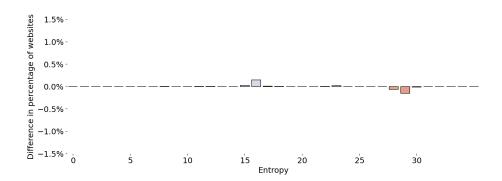
Figure 9: Change in entropy distribution when blocking scripts from known invasive fingerprinters (Disconnect, 2020).
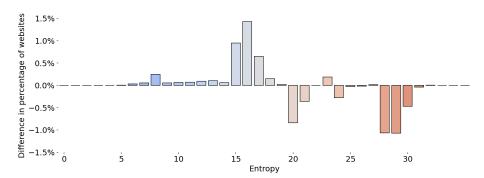

Figure 10: Change in entropy distribution when blocking scripts containing a fingerprinting signature (Acar et al., 2014).

surfaces, included only a small percentage of Chrome clients in the study, and capped the amount of information reported per client. All of these limits may have biased our results in ways that are difficult to quantify. For example, we only collected data from Chrome users who were logged in and consented to sharing their data with Google, a user population that may differ significantly from the overall population of the web.

Our study focused on measuring entropy, and while entropy is a reasonable proxy for fingerprinting risk (or, in other words, for the potential for fingerprinting), a website that collects high-entropy information is not necessarily a cause for user concern. The website may not be using the information for constructing fingerprints, or may be using the fingerprints for a benign purpose, such as preventing fraud or abuse. On the other hand, a website that collects low-entropy information may nonetheless be engaged in covert cross-site tracking via a non-fingerprinting method, such as link decoration or pixel tags.

Even our entropy estimates may be distorted, since they are based on the Chow-Liu decomposition, which provides an upper bound on entropy, but not a

lower bound. As we discussed in Section 4, this limitation is somewhat unavoidable, as computing more accurate entropy estimates would require a prohibitive number of samples. More technically, using a decomposition to estimate entropy while collecting data in a passive manner may have biased our estimates in a subtle way, since each term in the decomposition might have been estimated on a different set of clients.

8   Conclusions

Browser developers have proposed or implemented several mechanisms that try to address browser fingerprinting, including masking, fuzzing or coarsening the information the browser provides to websites (e.g., Brave, Firefox and Safari) and offering users the option to block domains known to engage in fingerprinting (e.g., Firefox and Edge). However, it remains unclear the extent to which such approaches can limit fingerprinting risks in-the-wild. Our measurement study is, to the best of our knowledge, the first effort to realistically quantify the amount of user-identifying information collected by websites that account for a large fraction of traffic on the Internet. The results of our study are useful for evaluating the impact of anti-fingerprinting methods, as well as for guiding the design of new anti-fingerprinting proposals.

References

Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 674–689, 2014.

Rohit Agrawal. Finite-sample concentration of the multinomial in relative entropy. *IEEE Trans. Inf. Theory*, 66(10):6297–6302, 2020.

Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. Fpselect: low-cost browser fingerprints for mitigating dictionary attacks against web authentication mechanisms. In *Annual Computer Security Applications Conference*, pages 627–642, 2020.

Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. "guess who?" Large-Scale Data-Centric study of the adequacy of browser fingerprints for web authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 161–172. Springer International Publishing, 2021a.

Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. A large-scale empirical analysis of browser fingerprints properties for web authentication. *ACM Transactions on the Web (TWEB)*, 16(1):1–62, 2021b.

András Antos and Ioannis Kontoyiannis. Convergence properties of functional estimates for discrete distributions. *Random Structures & Algorithms*, 19 (3-4):163–193, 2001.

Tomasz Bujlow, Valentín Carela-Español, Josep Sole-Pareta, and Pere Barlet-Ros. A survey on web tracking: Mechanisms, implications, and defenses. *Proceedings of the IEEE*, 105(8):1476–1510, 2017.

Siu-On Chan, Ilias Diakonikolas, Paul Valiant, and Gregory Valiant. Optimal algorithms for testing closeness of discrete distributions. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1193–1203. SIAM, 2014.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

Google Chrome. Google Chrome Privacy Notice. `https://www.google.com/chrome/privacy/`, 2022. Last accessed: 2023-06-28.

Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, 1969.

Disconnect. Our new approach to address the rise of fingerprinting, jul 2020. URL `https://blog.disconnect.me/our-new-approach-to-address-the-rise-of-fingerprinting/`. Invasive fingerprinting list retrieved on 2023-11-06.

Sam Dutton. Topics api overview, aug 2023. URL `https://developer.chrome.com/en/docs/privacy-sandbox/topics/overview/`.

Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings 10*, pages 1–18. Springer, 2010.

Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster analysis*. John Wiley & Sons, 2011.

David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*, pages 107–124. Springer, 2015.

Erik Flood and Joel Karlsson. Browser fingerprinting. *Master*.

Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 world wide web conference*, pages 309–318, 2018.

Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 878–894. IEEE, 2016.

Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Trans. Web*, 14(2):1–33, April 2020.

Song Li and Yinzhi Cao. Who touched my browser fingerprint? a large-scale measurement study and classification of fingerprint dynamics. In *Proceedings of the ACM Internet Measurement Conference*, pages 370–385, 2020.

João Pedro Figueiredo Correia Rijo Mendes. nophish–anti-phishing system using browser fingerprinting, 2011. `https://estagios.dei.uc.pt/cursos/mei/relatorios-de-estagio/?id=279`.

Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP*, 2012, 2012.

Liam Paninski. Estimation of entropy and mutual information. *Neural Comput.*, 15(6):1191–1253, 2003.

Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156*, 2018. Ranking retrieved on 2023-08-25.

Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. Long-term observation on browser fingerprinting: Users' trackability and perspective. *Proc. Priv. Enhancing Technol.*, 2020(2):558–577, 2020.

Kazuhisa Tanabe, Ryohei Hosoya, and Takamichi Saito. Combining features in browser fingerprinting. In *Advances on Broadband and Wireless Computing, Communication and Applications*, pages 671–681. Springer International Publishing, 2019.

Yihong Wu and Pengkun Yang. Minimax rates of entropy estimation on large alphabets via best polynomial approximation. *IEEE Transactions on Information Theory*, 62(6):3702–3720, 2016.

Desheng Zhang, Jianhui Zhang, Youjun Bu, Bo Chen, Chongxin Sun, Tianyu Wang, et al. A survey of browser fingerprint research and application. *Wireless Communications and Mobile Computing*, 2022, 2022.

A   Sample complexity of entropy estimation

The Chow-Liu (CL) upper bound depends on the entropy of marginal distributions $H(X_s)$ and on the mutual information between marginal distributions $I(X_s; X_{s'})$. The mutual information for discrete distributions can be further decomposed as

$$I(X_s; X_{s'}) = H(X_s) + H(X_{s'}) - H(X_s, X_{s'}) \tag{2}$$

which implies that computing the CL bound boils down to estimating entropy of marginal and pairwise marginal distributions. Therefore we will focus on estimating the entropy of discrete random variable X over domain $\{1, \ldots, k\}$ and its confidence interval which readily implies a confidence interval for mutual information and thus for the CL upper bound.

A.1   Entropy estimation

Assume that we are given n realizations, denoted by $x_1, \ldots, x_n$, from X with parameters $\mathbf{p} = (p_1, \ldots, p_k)$. This set of observation can be used to obtain a plug-in estimator $X^{(n)}$ of the distribution of X as

$$\widehat{p}_i^{(n)} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{1}_{x_j = i} \ .$$

In addition, the plug-in estimator can be naturally used to estimate the entropy as $H(X^{(n)})$.

Theorem 1. *Using the notation above, for any $\epsilon > \frac{2(k-1)}{n}$ and $n \in \mathbb{N}_+$, it holds that*

$$\Pr\left(H(X) - H(X^{(n)}) > \epsilon\right) \leq \exp\left(-\frac{\epsilon^2 n^2}{2 \sum_{i=1}^{k} \log^2 p_i}\right)$$
$$+ \exp\left(-\frac{n\epsilon}{2}\right) \cdot \left(\frac{e\epsilon n}{2(k-1)}\right)^{k-1}$$

and for any $\epsilon > 0$ and $n \in \mathbb{N}_+$, it holds that

$$\Pr\left(H(X^{(n)}) - H(X) > \epsilon\right) \leq \exp\left(-\frac{2\epsilon^2 n^2}{\sum_{i=1}^{k} \log^2 p_i}\right) \ .$$

Proof. Based on (4.1) of Paninski (2003), we can rewrite the ML estimate of entropy as

$$H(X^{(n)}) = H(X) + \sum_{i=1}^{k} \left(p_i - \widehat{p}_i^{(n)}\right) \log p_i - d_{\mathrm{KL}}\left(\widehat{\mathbf{p}}^{(n)}, \mathbf{p}\right)$$

23

which implies that

$$\Pr\left(H(X) - H(X^{(n)}) > \epsilon\right) \tag{3}$$

$$= \Pr\left(d_{\mathrm{KL}}\left(\widehat{\mathbf{p}}^{(n)}, \mathbf{p}\right) - \sum_{i=1}^{k}\left(p_i - \widehat{p}_i^{(n)}\right)\log p_i > \epsilon\right)$$

$$\leq \Pr\left(d_{\mathrm{KL}}\left(\widehat{\mathbf{p}}^{(n)}, \mathbf{p}\right) > \epsilon/2\right)$$

$$+ \Pr\left(\sum_{i=1}^{k} p_i - \widehat{p}_i^{(n)}\log p_i < -\epsilon/2\right) \tag{4}$$

which follows from the union bound. The first term of (4) can upper bound by Theorem 2. Using McDiarmid's inequality, see for example Proposition 1 of Antos and Kontoyiannis (2001), it holds that

$$\Pr\left(\sum_{i=1}^{k} p_i - \widehat{p}_i^{(n)}\log p_i < -\epsilon/2\right) \leq \exp\left(-\frac{\epsilon^2 n^2}{2\sum_{i=1}^{k}\log^2 p_i}\right)$$

since the L1 sensitivity of $\sum_{i=1}^{k} p_i - \widehat{p}_i^{(n)}\log p_i$ is $\frac{\log p_i}{n}$ for all $i \in [k]$ and n. On the other side, we have

$$\Pr\left(H(X^{(n)}) - H(X) > \epsilon\right) \leq \Pr\left(\sum_{i=1}^{k} p_i - \widehat{p}_i^{(n)}\log p_i > \epsilon\right)$$

which concludes the proof. $\qquad\square$

Theorem 2 (Theorem 1.2 of Agrawal (2020))**.** *Using the notation defined above, for all $\epsilon > \frac{k-1}{n}$, it holds that*

$$\Pr\left(d_{KL}\left(\widehat{\mathbf{p}}^{(n)}, \mathbf{p}\right) \geq \alpha\right) \leq e^{-n\alpha}\left(\frac{e\alpha n}{k-1}\right)^{k-1}$$

where

$$d_{KL}(\mathbf{p}, \mathbf{p}') = \sum_{i=1}^{k} p_i \log \frac{p_i}{p_i'} \ .$$

One can reformulate Theorem 1 as follows.

Corollary 1. *For any $n > 0$ and $\delta \in (0,1]$, it holds with probability at least $1 - \delta$ that*

$$H(X) - H(X^{(n)}) \leq \log\left(1 + \frac{k-1}{n}\right) + \sqrt{\frac{2\log(2/\delta)\log n}{n}}$$

and

$$H(X^{(n)}) - H(X) \leq \sqrt{\frac{2\log(2/\delta)\log n}{n}}$$

24

A.2 Estimating mutual information

Mutual information between discrete random variables can be decomposed into entropy values as it is given in (2). This result and Corollary 1 readily implies the following result.

Theorem 3. *Suppose a joint distribution (X_1, X_2) over domain $[k_1] \times [k_2]$ for which the marginal distributions X_1 and X_2 have parameters $\mathbf{p}_1$ and $\mathbf{p}_2$. Then, for any $\delta > 0$ and $n \in \mathbb{N}_+$, it holds that*

$$I(X_1; X_2) - I\left(X_1^{(n)}; X_2^{(n)}\right) \leq$$

$$\log\left(1 + \frac{k_1 - 1}{n}\right) + \log\left(1 + \frac{k_2 - 1}{n}\right) + 3\sqrt{\frac{6\log 2/\delta \log n}{n}}$$

and

$$I\left(X_1^{(n)}; X_2^{(n)}\right) - I(X_1; X_2) \leq$$

$$\log\left(1 + \frac{k_1 \cdot k_2 - 1}{n}\right) + 3\sqrt{\frac{6\log 2/\delta \log n}{n}}$$

Based on Theorem 3, note that the confidence interval for mutual information consists of the upper bound of the bias term and the statistical error. The statistical error becomes smaller than 0.5 with $n = 30000$ and $\delta = 0.05$. The bias term also becomes smaller than 0.5 when $n > 3k_1 \cdot k_2$. Thus if our goal is to achieve an 1-bit additive error of the mutual information with $\delta = 0.05$, we need

$$n = \max\left(3k_1 \cdot k_2, 30000\right)$$

samples. So if the domain size small, i.e. $k_1 \cdot k_2 < 10000$, additive error is dominated by statistical error term, and the domain size is larger than we need more sample to control the magnitude of the bias.

A.3 Estimating domain sizes

Note that the confidence intervals for the entropy and mutual information of random variables given in Sections A.1 and A.2 depend on their domain size (denoted k, k_1 and k_2 in the corresponding expressions). When each random variable represents a surface, its domain size is the number of distinct values for the surface among all clients. Since this information is unknown for most surfaces, we substituted the number of *observed* surface values instead. However, this quantity can be an underestimate for the true number of distinct surface values when the sample size is small, so we only used it when the number of samples exceeded the number of observed surface values by a factor of at least 30, a conservative heuristic. Otherwise we omitted all the mutual information terms associated with the surface when computing the Chow-Liu upper bound (Eq. (1)), which loosens the upper bound but does not impact its validity.

B   Minimizing observable impact of the experiment

When estimating the fingerprinting risk posed by Web APIs, it can happen that the estimation process itself creates a novel fingerprinting surface. For example, as we explained in Section 5, the second and third phases of our experiment assigned a different list of surfaces L_c to each client c. If our experiment had caused clients to record the values of surfaces in L_c while leaving the values of surfaces not in L_c unrecorded, then a website might be able infer L_c by measuring which surfaces take slightly longer to execute on client c then expected. Since L_c varies from client to client, it represents a fingerprinting risk. We mitigated this risk by having each client record the value of every surface in the experiment, whether or not the surface was contained in L_c. The recorded surface values were later filtered to only include the surfaces in L_c, and then asynchronously reported by the client to a server.

C   Special handling of codec, canvas readback and font surfaces

Since each surface consists of a Web API and possible input parameters, some Web APIs encompass so many surfaces that it is impractical to estimate their entropy even using the Chow-Liu decomposition (Eq. (1)). Instead we took a tailored approach to each of these cases. For media codecs, we found that nearly every surface with non-zero entropy was highly correlated with the platform of the device (*i.e.*, Android, Windows, *etc.*), and so platform entropy was a reasonable proxy for their joint entropy. For canvas readback surfaces, we estimated the entropy of each of the 50 most common canvas readback operations, and conservatively used the highest of these entropies as a proxy for the entropy of any other canvas readback operation. For fonts surfaces, we used a formula inspired by previous work (Eckersley, 2010; Fifield and Egelman, 2015).

D   Greedy surface subset assignment

As explained in Section 5, in the third phase of our experiment we assigned to each client c a subset of surfaces L_c, with the goal of collecting a sufficient number of samples from all pairs of the 60 'core' surfaces. We designed the assignment to satisfy two constraints. First, each surface pair $\{i, j\}$ had to be assigned to at least n_{ij} clients, where n_{ij} is the minimum number of samples needed to accurately estimate the mutual information between surfaces i and j, as determined by the derivation is Appendix A.2. Second, the joint entropy of the surfaces in any subset could not exceed a given threshold. We conservatively estimated the joint entropy of a subset using the sum of the marginal entropies of the surfaces in the subset, where each marginal entropy was estimated from data collected in the second phase of the experiment.

We used Algorithm 1 below to construct an assignment that respects the two constraints described above while also attempting to minimize the number of clients in this phase of the experiment. In each iteration, the algorithm greedily selects a new surface subset L^* that satisfies the sample requirements for surface pairs that were not met by any previously selected subset, while also respecting the joint entropy constraint. The subset L^* is then assigned to some clients that were not assigned a subset in a previous iteration.

Algorithm 1 Greedy surface subset assignment

Given:

- Number of samples n_{ij} needed for each surface pair $\{i, j\}$.
- Marginal entropy h_i of each surface i.
- Upper bound H on joint entropy of any surface subset.

Define for each surface subset L:

- Let $h(L) = \sum_{i \in L} h_i$ be the sum of marginal entropies of the surfaces in L.
- Let $s(L) = \sum_{\{i,j\} \subseteq L} n_{ij}$ be the total number of samples needed for all surface pairs in L.
- Let $m(L) = \max_{\{i,j\} \subseteq L} n_{ij}$ be the maximum number of samples needed for any surface pair in L.

1: **while** at least one of the n_{ij}'s is non-zero **do**
2:     Find a surface subset L^* that solves

$$\max_{L:\ h(L) \leq H} s(L).$$

3:     Select $m(L^*)$ clients that do not have a subset assignment.
4:     For each selected client c let $L_c = L^*$.
5:     For each $\{i, j\} \subseteq L^*$ let $n_{ij} = 0$.
6: **end while**

E   Replacing missing mutual information terms

Recall from Section 5 that if we do not have enough samples of a pair of surfaces (X_i, X_j) then we omit the mutual information term $I(X_i; X_j)$ from the right-hand side of the Chow-Liu decomposition (Eq. (1)). We are allowed to do this because mutual information is always at least zero. Alternatively, we can replace $I(X_i; X_j)$ with any valid lower bound. The next theorem gives a lower bound for $I(X_1; X_n)$ that can be significantly larger than zero provided that we have enough samples of the pairs $(X_1, X_2), (X_2, X_3), \ldots, (X_{n-1}, X_n)$.

Theorem 4. *For any random variables $X_1, \ldots, X_n$*

$$I(X_1; X_n) \geq \sum_{i=1}^{n-1} I(X_i; X_{i+1}) - \sum_{i=2}^{n-1} H(X_i).$$

Proof. By the Chow-Liu decomposition (Eq. (1))

$$H(X_1, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i) - \sum_{i=1}^{n-1} I(X_i; X_{i+1})$$

and thus by rearranging

$$H(X_1) + H(X_n) - H(X_1, \ldots, X_n) \geq \sum_{i=1}^{n-1} I(X_i; X_{i+1}) - \sum_{i=2}^{n-1} H(X_i).$$

The theorem follows by noting that $H(X_1, \ldots, X_n) \geq H(X_1, X_n)$ and $I(X_1; X_n) = H(X_1) + H(X_n) - H(X_1, X_n)$. $\square$

F Web APIs in the experiment

The following 161 Web APIs were selected for our experiment according to the criteria described in Section 5.

```
1  AudioBuffer.getChannelData
2  AudioContext.baseLatency
3  AudioDecoder.configure
4  BaseAudioContext.sampleRate
5  Document.plugins
6  FeaturePolicy.features
7  Gamepad.buttons
8  Gamepad.id
9  HTMLCanvasElement.getContext
10  HTMLMediaElement.canPlayType
11  InputDeviceCapabilities.firesTouchEvents
12  Keyboard.getLayoutMap
13  MediaCapabilities.decodingInfo
14  MediaCapabilities.encodingInfo
15  MediaDeviceInfo.kind
16  MediaDeviceInfo.label
17  MediaDeviceInfo.toJSON
18  MediaDevices.enumerateDevices
19  MediaDevices.getDisplayMedia
20  MediaDevices.getSupportedConstraints
21  MediaDevices.getUserMedia
22  MediaRecorder.mimeType
23  Navigator.appVersion
24  Navigator.bluetooth
25  Navigator.canShare
26  Navigator.cookieEnabled
27  Navigator.deviceMemory
28  Navigator.doNotTrack
```

```
29  Navigator.getBattery
30  Navigator.getGamepads
31  Navigator.getUserAgent
32  Navigator.getUserMedia
33  Navigator.hardwareConcurrency
34  Navigator.javaEnabled
35  Navigator.language
36  Navigator.languages
37  Navigator.maxTouchPoints
38  Navigator.mediaCapabilities
39  Navigator.mediaDevices
40  Navigator.mimeTypes
41  Navigator.platform
42  Navigator.plugins
43  Navigator.productSub
44  Navigator.userAgent
45  Navigator.userAgentData
46  Navigator.vendor
47  Navigator.webkitGetUserMedia
48  NetworkInformation.saveData
49  PaintWorkletGlobalScope.devicePixelRatio
50  Plugin.description
51  Plugin.filename
52  Plugin.name
53  RTCIceCandidate.address
54  RTCIceCandidate.candidate
55  RTCIceCandidate.port
56  RTCIceCandidate.relatedAddress
57  RTCIceCandidate.relatedPort
58  RTCPeerConnection.getReceivers
59  RTCRtpReceiver.getCapabilities
60  RTCRtpSender.getCapabilities
61  Screen.availHeight
62  Screen.availLeft
63  Screen.availTop
64  Screen.availWidth
65  Screen.colorDepth
66  Screen.height
67  Screen.id
68  Screen.internal
69  Screen.left
70  Screen.name
71  Screen.pixelDepth
72  Screen.primary
73  Screen.scaleFactor
74  Screen.top
75  Screen.touchSupport
76  Screen.width
77  ScreenOrientation.angle
78  ScreenOrientation.lock
79  ScreenOrientation.type
80  VisualViewport.height
81  VisualViewport.offsetLeft
82  VisualViewport.offsetTop
83  VisualViewport.pageLeft
84  VisualViewport.pageTop
85  VisualViewport.scale
```

```
 86  VisualViewport.width
 87  WebGLCompressedTextureASTC.getSupportedProfiles
 88  WebGLRenderingContext.getBufferParameter
 89  WebGLRenderingContext.getExtension
 90  WebGLRenderingContext.getParameter
 91  WebGLRenderingContext.getRenderbufferParameter
 92  WebGLRenderingContext.getShaderPrecisionFormat
 93  WebGLRenderingContext.getSupportedExtensions
 94  WebGL2ComputeRenderingContext.getBufferParameter
 95  WebGL2ComputeRenderingContext.getExtension
 96  WebGL2ComputeRenderingContext.getFramebufferAttachmentParameter
 97  WebGL2ComputeRenderingContext.getIndexedParameter
 98  WebGL2ComputeRenderingContext.getInternalformatParameter
 99  WebGL2ComputeRenderingContext.getParameter
100  WebGL2ComputeRenderingContext.getRenderbufferParameter
101  WebGL2ComputeRenderingContext.getShaderPrecisionFormat
102  WebGL2ComputeRenderingContext.getSupportedExtensions
103  WebGL2RenderingContext.getBufferParameter
104  WebGL2RenderingContext.getExtension
105  WebGL2RenderingContext.getIndexedParameter
106  WebGL2RenderingContext.getInternalformatParameter
107  WebGL2RenderingContext.getParameter
108  WebGL2RenderingContext.getRenderbufferParameter
109  WebGL2RenderingContext.getShaderPrecisionFormat
110  WebGL2RenderingContext.getSupportedExtensions
111  WebGL2RenderingContext.makeXRCompatible
112  WheelEvent.deltaMode
113  WheelEvent.deltaX
114  WheelEvent.deltaY
115  WheelEvent.deltaZ
116  Window.devicePixelRatio
117  Window.innerHeight
118  Window.innerWidth
119  Window.matchMedia
120  Window.name
121  Window.onbeforeunload
122  Window.ondevicemotion
123  Window.ondeviceorientation
124  Window.ondeviceorientationabsolute
125  Window.onlanguagechange
126  Window.onmessage
127  Window.onoffline
128  Window.ononline
129  Window.onorientationchange
130  Window.onscreenschange
131  Window.onsuspend
132  Window.ontimeupdate
133  Window.ontimezonechange
134  Window.orientation
135  Window.outerHeight
136  Window.outerWidth
137  Window.pageXOffset
138  Window.pageYOffset
139  Window.postMessage
140  Window.postMessage
141  Window.screenLeft
142  Window.screenTop
```

```
143  Window.screenX
144  Window.screenY
145  Window.scrollX
146  Window.scrollY
147  WorkerGlobalScope.onlanguagechange
148  WorkerGlobalScope.ontimezonechange
149  WorkerNavigator.appVersion
150  WorkerNavigator.deviceMemory
151  WorkerNavigator.hardwareConcurrency
152  WorkerNavigator.language
153  WorkerNavigator.languages
154  WorkerNavigator.mediaCapabilities
155  WorkerNavigator.platform
156  WorkerNavigator.serial
157  WorkerNavigator.usb
158  WorkerNavigator.userAgent
159  WorkerNavigator.userAgentData
160  CSS Media Features
161  Fonts
```