Unveiling the Impact of User-Agent Reduction and Client Hints: A Measurement Study

Asuman Senol
COSIC, KU Leuven
Leuven, Belgium
asuman.senol@esat.kuleuven.be

Gunes Acar
Radboud University
Nijmegen, The Netherlands
g.acar@cs.ru.nl

ABSTRACT

The user-agent string contains the details of a user's device, browser and platform. Prior work on browser fingerprinting showed that the user-agent string can facilitate covert fingerprinting and tracking of users. In order to address these privacy concerns, browsers including Chrome recently *reduced* the user-agent string to make it less identifying. Simultaneously, Chrome introduced several highly identifying (or high-entropy) user-agent *client hints* (UA-CH) to allow access to browser properties that are redacted from the user-agent string. In this empirical study, we attempt to characterize the effects of these major changes through a large-scale web measurement on the top 100K websites. Using an instrumented crawler, we quantify access to high-entropy browser features through UA-CH HTTP headers and the JavaScript API. We measure access delegation to third parties and investigate whether the new client hints are already used by tracking, advertising and browser fingerprinting scripts. Our results show that high-entropy UA-CHs are accessed by one or more scripts on 59.2% of the successfully visited sites and 93.8% of these calls were made by tracking and advertising-related scripts—primarily by those owned by Google. Overall, we find that scripts from ~9K distinct registrable (eTLD+1) third-party domains take advantage of their unfettered access and retrieve the high-entropy UA-CHs. We find that on 91.6% of the sites where high-entropy client hints are accessed via the JavaScript API, the high-entropy hints are exfiltrated by a tracker script to a remote server. Turning to high-entropy UA-CHs sent in the HTTP headers—which require *opt-in* or delegation—we found very limited use. Only 1.3% of the websites use the `Accept-CH` header to receive high-entropy UA-CHs; and an even smaller fraction of websites (0.4%) delegate high-entropy hints to third-party domains. Overall, our findings indicate that user-agent reduction efforts were effective in minimizing the passive collection of identifying browser features, but third-party tracking and advertising scripts continue to enjoy their unfettered access.

CCS CONCEPTS

• **Security and privacy → Web application security**; • **General and reference → Measurement**.

KEYWORDS

user-agent string; client hints; web privacy; fingerprinting; online tracking

ACM Reference Format:
Asuman Senol and Gunes Acar. 2023. Unveiling the Impact of User-Agent Reduction and Client Hints: A Measurement Study. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society (WPES '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3603216.3624965

WPES '23, November 26, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0235-8/23/11…$15.00
https://doi.org/10.1145/3603216.3624965

1 INTRODUCTION

The user-agent (UA) HTTP header was introduced in 1992 for "statistical purposes and the tracing of protocol violations" [1, 85]. The standard defined the UA header as a string that contains one or more tokens of software name and versions such as `LII-Cello/1.0 libwww/2.5`. The modern UA string typically contains far more extensive information about a user's browser, device and platform, and it is used for various purposes including analytics, debugging, content adaptation, or detecting incompatible, outdated or vulnerable browsers [82]. In addition to these legitimate use cases, the UA string may also enable stealthy cross-site tracking through browser fingerprinting, if combined with other device and browser information such as screen dimensions, installed fonts, or graphics capabilities [64]. A flavor of browser fingerprinting called *passive fingerprinting* only makes use of features that can be extracted from the received network packets, without running any client-side code [7, 81]. The information available to a passive fingerprinter include the user-agent string, IP address, `Accept` headers, clock skew [76] and protocol quirks [103]. Using passive fingerprinting, a third-party tracker may potentially link a user's web visits without running any client-side code, thwarting any detection efforts by the user or by web privacy measurement studies.

Empirical research on detecting passive fingerprinting on the web is almost non-existent, likely due to the elusive character of passive fingerprinting. Marketing materials of advertising companies, however, hint at the active use of user-agent data for tracking and ad targeting. For instance, TikTok recommends that websites send both the IP address and UA string of visitors through their server-to-server TikTok Events API for better ad targeting [16]. Mobile analytics and marketing company AppsFlyer's bulletin on Chrome's UA reduction mentions that user-agent data is used for "attribution via probabilistic modeling"[1] [19]. AppsFlyer recommends [19] their partner ad networks to collect and share with them OS (platform) version and device model user-agent client hints—two client

[1]Probabilistic attribution refers to attributing ad clicks, impressions or app installs without relying on deterministic identifiers such as AdID (Android), IDFA (iOS), or cookies.

Figure 1: User-agent reduction in desktop and mobile Chrome browsers.

hints that we found to be accessed most in our measurements (§4.2). Further, with the announced third-party cookie phase-out by Chrome [39] and anti-tracking measures turned on by default in Safari and Firefox, fingerprinting will likely be more crucial for cross-site tracking of users [73].

```
Mozilla/5.0 (Linux; U; Android 7.0; es-us; SM-A510M Build/NRD90M)
      AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome
      /61.0.3163.128 Mobile Safari/537.36 XiaoMi/Mint Browser/3.7.2
```

Listing 1: An example of a user-agent string (source: https://user-agents.net/s/rx5HpTAQIN).

Beyond its uses as a passive fingerprinting vector, the information contained in the user-agent string itself can be used for ad targeting and suppression. For instance, Google's documentation for real-time bidding (RTB) mentions that the user-agent header "has historically been included in bid requests to provide useful targeting data such as the initiating device's browser and platform" [52]. Using the UA string, advertisers may choose to show luxury product ads to high-end smartphone/laptop owners, or vice-versa: they could exclude users with outdated and budget devices from their ad campaigns [83]. The user agent string may also contain the primary language or locale of the device, which may reveal a user's country-of-origin and their immigration background [28, 44]. Consider the user-agent string given in Listing 1, which belongs to a budget smartphone brand (Xiaomi) with a (now) outdated Android version, having `es-US` locale (Spanish/US). The UA string may give advertisers enough information to correctly or incorrectly add this user to an imaginary "Latinx on a limited budget" marketing segment [12], or exclude them from certain job ads by their race, as Facebook (now Meta) allowed advertisers to do in the past [74]. Furthermore, knowing a user's country-of-origin may allow targeting ads appealing to (or exploiting) their cultural sensibilities [2, 17, 69].

In January 2020, Google announced plans to reduce the information conveyed in UA strings as part of their Privacy Sandbox project [41, 51, 99]. The stated objective of the effort was to minimize the identifying information in the UA string to prevent passive fingerprinting [51]. The plans also included the reduction of information in the associated `navigator` properties: `userAgent`, `appVersion`, and `platform` [50, 99]. The implementation of these changes occurred incrementally, starting with UA reduction for desktop browsers in May 2021, and ending with the reduction of Android Mobile (and Tablet) UA string and related JavaScript APIs in May 2023 [50]. Simultaneously, Google has introduced UA *client hints* (UA-CH) in Chrome, to expose the information reduced from the UA string in a structured manner. These new client hints can be accessed by first- or third-party scripts without any restrictions using a new JavaScript interface called `navigator.userAgentData`. In particular, `navigator.userAgentData.getHighEntropyValues` method returns high-entropy browser characteristics, including the unreduced UA string, device model and platform version. Adding these new client hints made it possible for web applications to easily access detailed UA information without parsing the complex UA string. For instance, Kline et al. report that the internal ruleset used by comScore to parse UA strings is "several thousand lines long" [75]. Thanks to the structured UA client hints web applications that rely on detailed browser information could retain their functionality. Making it easier to access high-entropy browser features may appear contradictory to the spirit of the UA reduction effort, but the stated objective of the UA reduction effort is to make *covert tracking* (via passive fingerprinting) more difficult, and active fingerprinting more transparent—*not more difficult* [88, 99]. Chrome achieves these dual objectives by reducing the information shared by default in the UA HTTP header, and requiring explicit JavaScript calls or HTTP/HTML-based *opt-in* to access high-entropy browser features. Making access to high-entropy browser properties auditable and actionable is motivated by Google's Privacy Budget proposal, which aimed to prevent active browser fingerprinting by limiting the entropy available to scripts [18, 40, 99][2].

In this empirical study, we investigate the effects of Google Chrome's UA reduction efforts on the exposure of potentially identifying browser features. The key contributions of our research are as follows[3]:

- We present a web measurement study of user-agent client hints (UA-CHs) on the home pages of the top 100,000 sites. Using an instrumented crawler, we capture and analyze JavaScript calls, HTTP headers, and HTML `<meta>` elements, which can be used to access, opt-in or delegate high- and low-entropy user-agent client hints.
- Monitoring the newly introduced JavaScript interface, we find that one or more third-party scripts access high-entropy client hints on 52,392 (58.4% of the successfully visited) websites. An overwhelming majority (93.9%) of these scripts are classified as advertising related. From those sites where we

[2]We note that the public GitHub repository of Privacy Budget proposal is last updated in late 2020 [40].
[3]The code and data from our study are available at https://github.com/ua-reduction.

Client Hint Header	Description	Example Value	Entropy
`Sec-CH-UA`	Browser name and major version	"Chromium";v="113", "Not-A.Brand"; v="24"	Low
`Sec-CH-UA-Mobile`	Boolean value indicating a mobile device	"?0"	Low
`Sec-CH-UA-Platform`	Operating system name	"Linux"	Low
`Sec-CH-UA-Full-Version` (Deprecated)	Unredacted UA version	"113.0.5672.63"	High
`Sec-CH-UA-Full-Version-List`	List of unredacted UA versions	"Chromium";v="113.0.5672.63", "Not-A.Brand";v="24.0.0.0"	High
`Sec-CH-UA-Platform-Version`	Operating system version	"NT 6.0", "5.15.0", or "17G"	High
`Sec-CH-UA-Arch`	Platform architecture	"ARM", or "x86"	High
`Sec-CH-UA-Model`	Device model	"Pixel 2 XL"	High
`Sec-CH-UA-Bitness`	CPU architecture bitness	"32" or "64"	High
`Sec-CH-UA-WoW64`	Whether the UA is a 32-bit binary running on 64-bit OS	?0 or ?1	High

Table 1: User-agent client hint headers along with their descriptions, sample values, and entropy levels.

observed third-party access, we observe third-parties exfiltrate high-entropy values to their endpoints on 47,691 (91.0%) sites.
- Using JavaScript instrumentation, we detect fingerprinting attempts by third-party scripts and find that only a minority of the fingerprinting scripts already take advantage of newly introduced high-entropy client hints.
- We examine the usage of high-entropy CH headers sent in HTTP requests and find that their use is scarce. This indicates that the UA reduction efforts succeeded in minimizing the exposure of high-entropy browser features to HTTP endpoints that may use them for passive fingerprinting.

2 BACKGROUND AND RELATED WORK

In the more than 30 years since its introduction, the user-agent string has grown into a long, detailed and somewhat confusing string that contains many ossified parts. Perhaps only a minority of tech professionals know what `KHTML` is, what is special about `AppleWebKit` version 537.36 [95], and why Chrome and Safari mention `Mozilla/5.0` in their UA string [3, 85]—despite having seen these strings potentially thousands of times. Parallel to this ossification, many new properties of the browser, operating system (OS) and physical device were added to the UA string, so web servers can send compatible and localized web content. In addition to legitimate use cases such as delivering optimized versions of images or videos for devices with different form factors [93], the UA string can also be used to obtain a potentially unique *browser fingerprint* by combining it with other information [59, 64, 70, 77, 84].

The UA string influences the uniqueness of a user's fingerprint due to the discriminating information it contains. In a seminal study by Eckersley [65], the UA string ranked as the third-most distinguishing browser feature after fonts and plugins. Specifically, by collecting the fingerprints of 470K volunteered participants, Eckersley found that the UA string contains 10.0 bits of Shannon entropy—which means only one in 1024 (2^{10}) other browsers is expected to share the same UA string for a given browser. Following a similar approach, a 2016 study by Laperdrix et al. [77] showed that UA string is the most identifying feature for mobile browsers and third for identifying features for desktop browsers. Surprisingly, one of every four smartphones in their dataset had a unique UA string—primarily due to the inclusion of details such as the device model, phone carrier, and firmware version. We note that while we focus on browsers as the primary software to access the web, mobile apps, smart devices and software libraries can also be classified as user agents, and have user-agent strings.

2.1 What changed and how?

In January 2020 Chrome developers announced their "[i]ntent to deprecate and freeze the UA string" [99]. While the initially announced timeline was not followed due to COVID, Chrome reduced or froze part of the UA string in several phases [50]. The milestones included replacing the minor version number with zeros and using a fixed Android version. By February 2023, the UA string for Android mobile and tablet devices was fully reduced [50]. Google's UA Reduction effort had three main components:

(1) Reducing the granularity of the UA string and freezing parts of it (Figure 6). The minor version number is replaced with zeros in Chrome 101 (June 2022). As an example, Chrome/101.3.2.1 would become Chrome/101.0.0.0 [55] (Figure 1). Further, CPU and platform-related details are simplified for the desktop browsers in Chrome 107 (February 2023), and the Android version number is replaced with the fixed string "10" (May 2023) [50].
(2) The second component involves the implementation of user-agent client hint (UA-CH) HTTP headers—a structured alternative to the UA string. UA-CH headers are enabled by default in Chrome 89, released in March 2021 [88].
(3) The third component introduces a new JavaScript interface called `NavigatorUAData`, which contains properties and methods for scripted access to UA-CHs (introduced in Chrome 90, released in April 2021) [34].

2.2 User-Agent Client Hints

User-agent client hints (UA-CHs) offer a structured alternative to retrieve the information redacted from the UA string. This client hints to complement the UA reduction efforts to maintain compatibility. A client hint is introduced for all UA string components, a complete list of which is given in Table 1. UA-CHs are categorized as low- and high-entropy, based on how identifying they are. For instance, *full-version* hint, which contains a detailed build version is categorized as high entropy; while *mobileness* hint indicating whether a device is mobile or not is categorized as low entropy.

The entropy category of a UA-CH determines whether the corresponding header is automatically appended to every request or not: low-entropy hints are sent in every request, while high-entropy hints require explicit opt-in or delegation. To the best of our knowledge, the division between the high- and low-entropy client hints is not based on publicly available research. We observe that rare values for low-entropy hints such as *platform=Linux* or *Fuchsia*, can be identified if combined with other browser features.

The list of user-agent client hints is not fixed. For instance, Chrome recently added a client hint representing the form factor of the user's device and the corresponding `Sec-CH-UA-Form-Factor` header [82]. This information was historically conveyed as a `<deviceCompat>` token in the user-agent string (e.g., "Tablet", "VR", etc.) and was added during our study. Other proposed user-agent client hints include a hint that indicates whether the browser needs an update [26] and whether the browser is a self-declared bot [43]. Note that UA-CHs are just one type of client hints; others include *Device Client Hints*, *Network Client Hints* and *User Preference Media Features Client Hints* [29].

During our investigation, we spotted a few mismatches between the latest UA-CH proposal [4] and its implementation in Chrome. For instance, since August 2021, the UA-CH standard requires an empty string to be returned in the *platform version* for Linux devices [32]. However, as of July 2023, Chrome still returns the Linux kernel version (e.g., 5.15.0) in the platform version client hint on Linux [96].

All the information redacted from the user-agent string can be obtained through high-entropy client hints. These hints can be accessed via HTTP headers, or by using the JavaScript API. Notably, the JavaScript API does not require any permission or opt-in from the website—even for the scripts running in third-party iframes. On the other hand, to receive high entropy UA-CHs in the HTTP headers even a first-party needs to opt-in. Detailed explanations of both methods are provided in the subsequent sections.

Access to User-Agent Client Hints via HTTP By default, Chrome appends three low-entropy client hints, i.e, *platform name*, *major browser version*, and *mobileness* in each HTTP request, if the connection is secure (i.e., HTTPS). On the other hand, high-entropy client hints require explicit opt-in or delegation to be sent to first, or third parties, respectively. Top-level origins may request high-entropy UA-CH headers by sending an `Accept-CH` response header as shown in Step 2 in Figure 2. `Accept-CH` can contain a list of CH header names such as `Sec-CH-UA-Platform-Version`. The browser then sends these additional CH headers in all subsequent requests to the first-party domains (Step 3 in Figure 2). Additionally, client hints can be requested at the connection time using HTTP/2 or HTTP/3 `Accept_CH` frames, or using the TLS Application-Layer Protocol Settings (ALPS) extension mechanism [21, 22, 45]. Requesting the client hints at the connection layer instructs the browser to append the UA-CHs to the first HTTP request, which saves a round trip at the application layer.

Servers can also use the `Critical-CH` response header when certain high-entropy UA hints are required on the first navigational request. Clients that receive a response with a `Critical-CH` header must check if high-entropy hints are sent in the original request. If not, the client will resend the original request with the requested hints (Step 3 in Figure 3).

[4]User-Agent Client Hints proposal is a Draft Community Group Report and it is still in the pre-standardization stage [47].

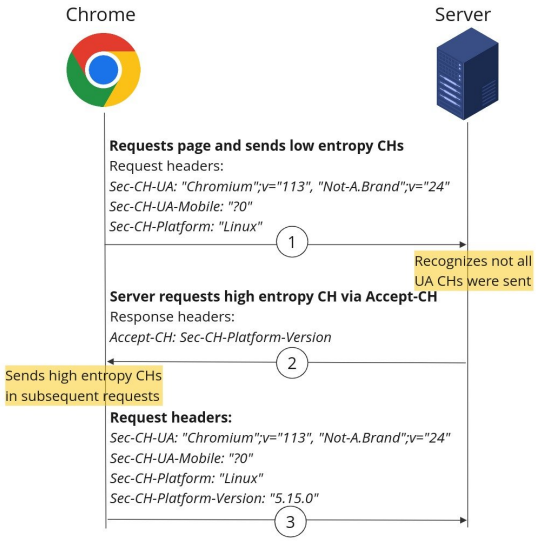

Figure 2: The `Accept-CH` response header mechanism used by servers to request high-entropy user-agent client hints from clients. After receiving an `Accept-CH` header during a secure top-level navigation request, the browser appends the requested client hint headers in subsequent requests. The image is adapted from Google's documentation [100].

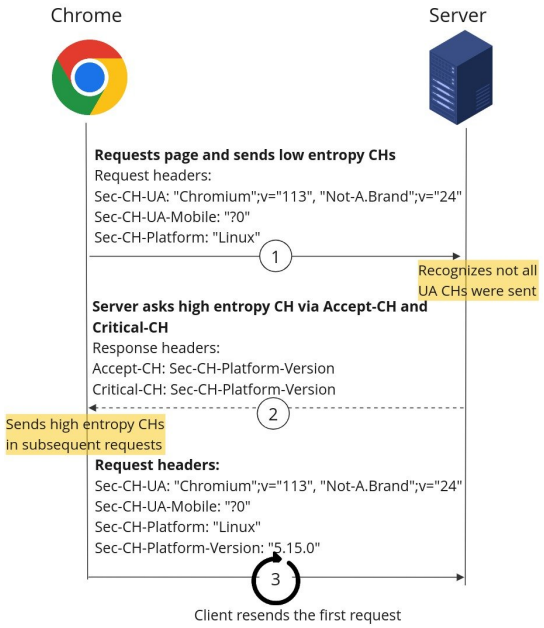

Figure 3: Requesting high-entropy CHs initially using `Critical-CH` header: When the server asks for a `Critical-CH`, the client will retry the initial webpage request with it. The image is adapted from Google's documentation [100].

Access to User-Agent Client Hints via the JavaScript API In the past, the UA information was exposed to scripts through the `userAgent`, `appVersion`, and `platform` properties of the `navigator` object. However, as part of UA reduction efforts, the information accessible through those properties was also reduced. In order to provide an alternative mechanism for scripts to access high-entropy browser features, Chrome introduced a JavaScript API for user-agent client hints [49]. This API includes an interface named `NavigatorUAData` (Appendix A), which provides low-entropy UA values, and an asynchronous method called `navigator.userAgentData.getHighEntropyValues`, offering high-entropy UA values [82]. We have provided an example of a `navigator.userAgentData` call and its corresponding output value in Appendix A. The `getHighEntropyValues` method allows client-side scripts to selectively retrieve high-entropy UA client hints such as `architecture`, `bitness`, `platformVersion`, `uaFullVersion` and `fullVersionList`[5]. This method offers a more explicit and controlled approach to retrieving browser features, allowing for better monitoring and transparency. Like UA-CH headers, `getHighEntropyValues` is restricted to HTTPS pages. An example call and output in Google Chrome are shown in the Listing 2.

```
navigator
  .userAgentData.getHighEntropyValues(
      ["brands", "mobile", "bitness", "platform",
      "platformVersion", "architecture",  "model",
      "uaFullVersion", "fullVersionList", "wow64"])
  .then((ua) => { console.log(ua) });
// OUTPUT:
{
    "architecture": "x86"
    "bitness": "64"
    "brands":[
        {
            "brand":" Not A;Brand",
            "version":"24"
        },
        {
            "brand":"Chromium",
            "version":"113"
        },
        {
            "brand":"Google Chrome",
            "version":"113"
        }
    ],
    "fullVersionList":[
        {
            "brand":" Not A;Brand",
            "version":"24.0.0.0"
        },
        {
            "brand":"Chromium",
            "version":"113.0.5672.63"
        },
        {
            "brand":"Google Chrome",
            "version":"113.0.5672.63"
        }
    ],
    "mobile": false,
    "model": "",
    "platform": "Linux",
    "platformVersion": "5.15.0",
    "uaFullVersion": "113.0.5672.63",
    "wow64": false
}
```

Listing 2: Sample output of the `getHighEntropyValues` in Google Chrome. returning requested user-agent client hints including the high-entropy ones.

[5]Note that the camelCase UA client hint names in JavaScript correspond to UA-CH headers listed in Table 1; e.g., `platformVersion` => `Sec-CH-UA-Platform-Version`.

Remarkably, any first- or third-party script can call the `getHighEntropyValues` method to access high-entropy UA-CHs without permission, opt-in, or delegation from the website. This is in stark contrast to how sending high-entropy UA-CH HTTP headers requires active opt-in even for the website (or the first-party domain) itself.

GREASE for User-Agent Brands To prevent reliance on specific UA strings in specific positions, the Generate Random Extensions and Sustain Extensibility (GREASE) technique [82] is adopted from TLS [57] to UA client hints. In the UA-CH context, GREASE involves introducing intentionally incorrect or random entries in the UA brands list to increase compatibility and prevent sites from blocking unknown browsers. Specifically, browsers are required to include multiple values in the brands list, with one being an arbitrary value. The order of values must change over time to avoid dependence on specific positions, and a random set of separators should be used [10]. Unfortunately, the recent bug reports by Vivaldi —a Chromium-based browser— show that GREASE does not ensure equal treatment [54, 102]. According to reports, websites already started parsing client hints with misplaced assumptions, and deny access to browsers with small market shares due to unexpected brand CH values [102].

2.3 Delegating hints to third-parties

By default, UA-CHs are not sent for third-party subresource requests, and third parties cannot request high-entropy UA-CHs themselves. Instead, the first-party server must send a `Permissions Policy` header that lists the origins that may receive the UA-CHs [6]. Examples of implementing this policy can be found in Figure 4.

Let's take an example where example.com embeds a resource from sample-cdn.com which needs the full version of the browser. The `Sec-CH-UA-Full-Version` hint can be requested by https://example.com, but it must be explicitly delegated to https://www.sample-cdn.com using the `Permissions-Policy` response header as shown in Step 3 in Figure 4. Then the request to subresources on sample-cdn.com (script source ⓑ in Figure 4) includes the delegated hint which is `Sec-CH-UA-Full-Version`: "113.0.5672.63".

Additionally, multiple hints can be defined for multiple origins. For instance, example.com delegates multiple hints to multiple origins with this response header as shown in Listing 3.

```
Accept-CH: Sec-CH-UA-Full-Version
Permissions-Policy: ch-ua-full-version=(self "https://www.sample-
      cdn.com"), ch-ua-model=(self "https://www.sample-cdn.com" "
      https://analytics.com")
```

Listing 3: Client hint delegation to multiple domains.

To enable the delegation of CHs to cross-origin iframes, it is essential to specify the CHs in the iframe's `allow` attribute, along with the `Permissions Policy` response header. Figure 4, presented illustrates this process in ⓓ. When an iframe with the source https://sample-cdn.com includes the `allow` attribute, it can receive the `Sec-CH-UA-Full-Version` CH header. However, as shown in ⓔ of Figure 4, the absence of the `allow` attribute results in the exclusion of this high-entropy CH.

[6]The `Permissions Policy` (previously, `Feature Policy`) header allows enabling and disabling browser features such as microphone and camera for all or certain origins.

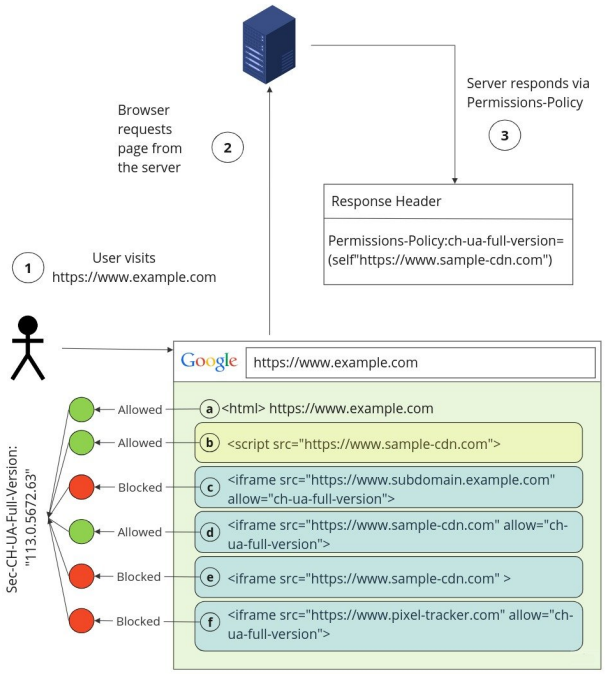

Figure 4: Delegating client hints to Third-Party Origins by `Permissions-Policy`. The image is based on a figure in Google's documentation [78].

Sample ⓒ in Figure 4 illustrates a same-site, cross-origin frame (subdomain.example.com). Despite having an iframe `allow` attribute for UA-CHs, the high-entropy hint will not be sent when loading this iframe, because it is not included in the origin list.

2.4 User-Agent Client Hint opt-in and delegation via HTML

Publishers who cannot modify their website's `Permissions Policy` HTTP header can use HTML to opt-in to high-entropy client hints or delegate them to third parties. The HTML `<meta>` element can be used to opt-in to high-entropy CHs for first-party requests. In this case, the `http-equiv="Accept-CH"` attribute, along with the `content` attribute can be used. The `content` attribute holds the list of hints to be delegated, along with the desired recipient origins. Similarly, the `http-equiv` attribute of the `<meta>` element enables explicit delegation of high-entropy CHs to third-party domains. To achieve this, the `http-equiv="delegate-ch"` attribute is used, along with the `content` attribute, which designates the hints to be delegated and the corresponding recipient origins.

2.5 Statuses and positions of other browser vendors

In this section, we summarize the status and viewpoints of Apple and Mozilla with respect to UA reduction and UA client hints. Due to space limitations we only provide a very high-level summary of each vendor's position and we do not claim that the following is an exhaustive description:

Mozilla Independent of Google's efforts, Mozilla had already frozen the rendering engine version (`rv`) in Firefox 110 to avoid being detected as IE11 [89]. Mozilla also reduced the information exposed in the UA string over time [8, 20]. With respect to introducing user-agent client hints, Mozilla initially held a negative stance. Mozilla initially labeled UA-CHs as *harmful* in their web standard positions [90], citing concerns such as additional fingerprinting risks, but they subsequently updated their position to *neutral* [33]. While there are bugs to track the implementation of UA-CHs [4, 13], no work has been done toward that implementation as of July 2023.

In their critique of UA-CHs, Mozilla highlights the bandwidth overhead of sending multiple low-entropy `Sec-CH` headers in each request, along with additional fields requested by sites using the `Accept-CH` mechanism [97]. Note that while HTTP/2 and HTTP/3 support header compression, even the compressed fields may consume a non-trivial amount of bandwidth. We leave the bandwidth overhead measurement of UA-CHs to future work.

Apple As early as 2017 Apple attempted to freeze the UA string to reduce compatibility issues and defend against browser fingerprinting [5, 6, 62]. Seemingly due to compatibility concerns, they later unfroze the major OS version [15]. Apple initially held a negative stance against UA-CHs and filed several issues on the public GitHub repository, which are addressed except for one as of July 2023 [11, 94]. The only remaining issue filed by Apple concerns the `getHighEntropyValues` method, which returns all requested high-entropy hints, including those that have not been explicitly opted in via the `Accept-CH` header [91]. This means, regardless of its origin and security context, a script can access all high-entropy UA client hints, even if the first-party domain did not send an `Accept-CH` header to opt-in to high-entropy client hints. Citing the W3C specification, Apple engineers suggested that prior opt-in must be required for accessing high-entropy hints from scripts [91]. Currently, the first party can *opt-out* from exposing high-entropy client hints to scripts from any origin by sending a restrictive `Permissions-Policy` header.

Note that Apple requires all browsers running on iOS to use its own WebKit rendering engine [60]. Thus, Apple's position towards client hints also determines the availability of UA-CHs on iOS versions of Chrome, Firefox and other browsers.

2.6 Related Work

The research community has expressed considerable interest in the UA string and its implications for web privacy and security. In this section, we discuss the most pertinent research in this domain.

In his pioneering study, Eckersley [64] demonstrated that it is feasible for websites to uniquely identify users at scale through their browser fingerprints. Collecting browser fingerprints of over 470K volunteers, Eckersley showed that over 94% of the users having Flash or Java have a unique fingerprint. Following Eckersley's lead, several studies introduced novel ways to fingerprint browsers such as Canvas, Canvas Font and Audio Context API fingerprinting, as well as WebRTC local IP discovery [67, 84]. In our study, we use heuristics proposed by Englehardt and Narayanan [67] to detect these four types of fingerprinting attempts. Vastel et al. [98] devised a technique to link evolving browser fingerprints. Analyzing 98,598 browser fingerprints from 1,905 distinct browser instances

collected over a two-year period, they found that fingerprints regularly change due to software updates, but their algorithm can link evolving browser fingerprints with reasonable accuracy for months.

A type of fingerprinting called *passive fingerprinting* involves extracting unique device fingerprints without running any client-side JavaScript code. Passive fingerprinting relies on information readily available to a web server such as IP address and HTTP headers, including `User-Agent`, `Accept-Language`, and `Accept` headers [7, 81]. These fingerprints can then be used to track and identify devices across different browsing sessions. The main privacy concern with passive fingerprinting is that it enables covert tracking since it is undetectable to any client-side analysis, including web measurement studies that use heavily-instrumented browsers [81]. An influential study by Kohno et al. presented a method to remotely identify a specific physical device using its *clock skews* [76]. Using timestamps sent in network packets, their method could identify devices that were geographically distant, connected via different access technologies, or behind firewalls/NATs. Analyzing web server logs of Hotmail and Bing services, Yen et al. [101] found that 60-70% of the distinct user-agent strings in their dataset were unique. Kline et al. analyzed a corpus of over one billion user-agent strings collected by comScore, a media traffic measurement and analytics company [75]. They found that less popular (*low-volume*) UA strings generate the overwhelming majority of traffic [75].

Variations in how different devices and software implement specific network protocols can lead to distinguishable patterns at the network layer. Zalewski implemented `p0f`, which passively identifies operating systems and software using differences in their protocol implementations [103]. `p0f` uses a variety of network and application-level features extracted from TCP, IP and HTTP headers including the layout and ordering of TCP options; syntax and order of HTTP headers; TCP timestamp increments; and various other implementation quirks. A remarkable feature of `p0f` is how little data it needs for fingerprinting: a single standard SYN packet could be enough to distinguish many systems. While `p0f` mainly uses protocol quirks for fingerprinting, it uses the user-agent string to detect mismatches between the declared and actually used software and OS, which may indicate *dishonest software* or manipulation by on-path proxies. Following a similar approach, several tools and libraries were developed for TLS fingerprinting [42, 46, 79]. Using differences in TLS versions, ciphersuites, supported extensions and compression options, these projects aimed to identify the TLS software or libraries for objectives such as detecting unpatched browsers, or malicious software that impersonates real browsers. Anderson and McGrew's research showed that precise OS version detection is possible by merging TLS features with TCP/IP and HTTP protocol data [56].

Another line of research and applications used UA strings to detect malware and compromised hosts. Grill et al. detected infected hosts by using the discrepancy between a user's actual UA string, and the UA string used by potential malware [68]. UA strings can also be used for detecting malicious and benign bots. The npm package "isbot", for instance, claims to distinguish between good bots such as web crawlers and bad bots that launch DDoS attacks and spam campaigns [30].

We contribute to the literature by performing a large-scale web measurement study focusing on the implications of user-agent reduction and the simultaneous introduction of potentially identifying user-agent client hints. Our study provides empirical data that can be used to evaluate the effectiveness of these major changes and their impact on web privacy.

3 METHODOLOGY

In this section, we describe the methods and tools developed to measure the usage of UA-CH HTTP headers and the JavaScript API.

3.1 Extending Tracker Radar Collector

In order to develop our crawler, we extended Tracker Radar Collector (TRC), a Puppeteer-based web crawler that records certain JavaScript API accesses, HTTP requests & responses, cookies, and other data related to web measurements. Since TRC only saves a pre-defined list of HTTP headers, we extended its allow-list by adding the ten UA-CH HTTP headers listed in Table 1, and other HTTP headers used to opt-in and delegate high-entropy CHs (e.g., `Accept-CH` and `Critical-CH`). In addition, we parsed the *meta* and `iframe` elements' attributes, as opt-in and delegation can also be accomplished through these means (§2.4). In order to monitor access to high-entropy UA values by scripts, we intercept calls to `navigator.userAgentData.getHighEntropyValues` and save the arguments and the call stack. We use the function arguments to identify the requested high-entropy client hints, and capturing the JavaScript call stack allows us to identify the caller script's URL. Additionally, we instrument fingerprinting-related method calls and property accesses using a separate collector (§3.2). For both `getHighEntropyValues` and fingerprinting detection, we override the relevant object's *getters* to intercept the function calls. While TRC has functionality for intercepting JavaScript API calls, we added a separate collector, because the existing collector misses the initial function calls due to a known TRC issue [23]. Moreover, we extended TRC's network instrumentation to save all WebSocket traffic and HTTP POST payloads to better examine whether any high-entropy values are being exfiltrated. This effort complemented the existing network instrumentation of TRC, which already intercepted GET requests.

Only visiting the webpage for a few seconds may not accurately represent an actual user's visit. To better simulate a real user, we scrolled to the bottom of the page and then scrolled up, waiting for 5 seconds before gathering the necessary data. During the initial phases of the study, Chrome in headless mode did not support client hints. While we initially planned to run our crawler in headful mode within a virtual display (e.g. via `Xvfb`), client hint support was added to headless Chrome in February 2023 [9].

We override the crawler's UA string and client hints using Puppeteer's *page.setUserAgent* [36] method. This method allows passing a `userAgent` string and structured UA metadata (`userAgentMetadata`) which we use to specify all possible UA-CHs. The UA values we use to override were taken from a real Chrome browser running on Linux, and are given in Appendix B.

3.2 Detecting fingerprinting attempts

To determine whether existing fingerprinting scripts are leveraging the newly introduced high-entropy client hints, our first step is to

identify these fingerprinting scripts. However, this can be a challenging task, as discerning the true intent of a script may be difficult. For instance, a script might use the AudioContext API solely for processing and synthesizing audio in web applications, without using the data for fingerprint creation. Previous research has employed either machine learning-based techniques [70] or heuristics [67] to detect such scripts. Due to the scope of our study, we adopted Englehardt and Narayanan's heuristics (see Appendix C) to detect fingerprinting scripts, focusing exclusively on four sub-types: Canvas, WebRTC, Canvas Font, or AudioContext fingerprinting [67]. Briefly explaining each fingerprinting method, Canvas fingerprinting involves using the HTML5 Canvas API to draw images with slight variations for user tracking [84]. WebRTC local IP discovery exploits the WebRTC API to obtain the local IP of the browser [67]. Canvas Font fingerprinting utilizes the Canvas API to render text in various fonts and styles, enabling the analysis of rendering differences to create distinct fingerprints [67]. Similarly, AudioContext API fingerprinting use signal processing features to extract a fingerprinting of the audio stack of the device [67]. To intercept function calls, return values and function arguments, we overrode the getter and setter methods of those Web APIs. The script responsible for overriding was injected into every page including their subframes immediately after the page was created, before the page's scripts run.

3.3 Interaction with consent dialogs

Since the GDPR came into effect in 2018, numerous websites started to display consent dialogs when accessed from the European Union (EU)—or even from the US, to some limited extent [87]. To fully measure the access to the high-entropy UA-CHs, we opted to accept all personal processing (or cookies). In order to handle consent dialogs accurately and automatically, we used code extracted from Priv-Accept [71, 80], a specialized crawler designed to accept consent dialogs on websites. Since Priv-Accept was only available in Python, we ported its consent interaction code to JavaScript and integrated it into our crawler. Essentially, our Priv-Accept (NodeJS) script searched for specific HTML elements such as `a`, `button`, `div`, `span`, `form`, `p`, containing keywords such as "Accept". Upon detecting such an *accept element*, Priv-Accept automatically clicked it to accept all personal data processing and cookies.

3.4 Detection of high-entropy value exfiltrations

Do the scripts that access high-entropy UA values send it to their servers or share it with other third parties? The high-entropy UA hints retrieved by the scripts may potentially be used for feature detection or other client-side optimization, which could alleviate privacy concerns since the high-entropy values may not leave the browser. In other cases, scripts that retrieve the high-entropy UA values via JavaScript calls may send them to several third-party domains, potentially for tracking and advertising purposes. Detecting these exfiltrations relies on identifying the presence of high-entropy UA values in the HTTP request payloads (excluding the `Sec-CH-*` headers). A potential challenge for accurately detecting such exfiltrations is to identify the sent high-entropy hints in encoded, hashed, or obfuscated payloads. Previous studies in web privacy measurement have taken various approaches to tackle this challenge. In order to detect Personally Identifiable Information (PII) exfiltrations from contact forms, Starov et al. [92] conducted a comparative analysis of HTTP parameters observed in multiple crawls. However, their method requires manual analysis and multiple crawls, where they provide different identifiers. Instead, we follow Englehardt et al.'s approach [66], which involves searching for multi-layered encodings and hashes (e.g., Base64, SHA-256) of search strings in the HTTP request URL, headers and POST data.

Naively searching for all high-entropy UA values in the HTTP traffic may lead to false positives. Consider the high-entropy hint `Sec-CH-UA-Bitness`, which may take the values of 32 and 64. These two-digit values may appear in the HTTP traffic due to unrelated reasons, for instance as part of random strings. In order to prevent such false positives, we only searched for two high-entropy client hints: the platform version (`5.15.0`) and the full UA version (`113.0.5672.63`). We begin by generating a precomputed pool containing all possible sets of tokens through iterative encodings and hashes of the platform version and full version. We then search for exfiltrations in the referrer header, URL, and POST bodies of the requests, using potential separator characters (e.g., '=') to split the content. We exhaustively apply all possible decodings and check if the decoded result matches any entry in the precomputed pool. This process is repeated until we reach a maximum of three layers of encodings or decodings. A detailed list of the encoding and hash algorithms used can be found in Appendix D.

3.5 Identifying tracking-related requests

To identify tracking-related requests, we employed the uBlock Origin Core npm package [27], which replicates the blocking behavior of the widely used uBlock Origin tracking protection extension [48]. By using the default filter lists employed by uBlock Origin, including EasyList [24] and EasyPrivacy [25], we determined the blocked status of each request. To achieve this, we provided uBlock Origin Core with the resource type of the request (e.g., image or script), as well as the page and request URL. When detecting whether a script is tracking-related or not, we use the script URL extracted from the JavaScript call stacks (§3.1).

3.6 The crawl

In June 2023, we crawled the homepages of the top 100K sites in the April 2023 Chrome User Experience Report list [72], excluding sites with identical fully qualified domain names. 91% (89,763 sites) of websites could be visited successfully while the remaining visits failed due to errors including DNS and timeout errors. Priv-Accept accepted personal data processing on around 21% of the visited sites, as revealed by the crawl logs. The crawl was performed using a cloud-based (DigitalOcean) server located in the United States (US). The crawl took four days, and it was run on a server equipped with 8 vCPU cores and 16GB of RAM. We chose a server located in the US, mainly to minimize the consent dialogs our crawler needs to deal with. While we use Priv-accept to detect and interact with consent dialogs (Section 3.3), certain consent dialogs may be missed by our crawler, leading to a bias in the measurement. We leave it to future work to perform a comparative study based on multiple vantage points.

4 MEASUREMENT RESULTS

Recall that user-agent client hints can be gathered using JavaScript calls and user-agent client hints HTTP headers. Below, we present separate measurements for each access method to conduct a comprehensive analysis.

4.1 `getHighEntropyValues` calls and exfiltrations

We observed a call to `getHighEntropyValues` method by one or more third-party scripts on a total of 52,392 unique sites, which amounts to 58.4% of the successfully visited sites. Table 2 gives an overview of the presence of these JavaScript API calls across the analyzed websites. We found that almost all calls (98.6%) are due to third-party and tracking-related scripts. According to Chrome usage metrics, `getHighEntropyValues` is invoked during 53.3% of the page loads [37]. We believe that our measurement (58.4%) is slightly higher because our study focused on the top 100K sites. Recall that we determine the caller script using the JavaScript stack traces observed during a call to `getHighEntropyValues`. In order to detect tracking-related scripts we use the labels returned by the uBlock Origin for the caller script's URL as explained in §3.5.

	All	Third party	Tracking related
`getHighEntropyValues` calls	53,148	52,392	51,630
Hi-ent. UA-CH exfiltrations	48,355	47,691	47,285

Table 2: The number of distinct websites where high-entropy UA-CHs were retrieved by a script and exfiltrated. On over 91% of the websites, scripts exfiltrate high-entropy UA-CHs that they retrieve via the `getHighEntropyValues` method.

We analyzed the categories of third-party scripts (obtained via inspecting JavaScript stack traces) that retrieve high-entropy client hints to understand the purposes of these API calls. For domain categorization, we relied on the Tracker Radar dataset [61], which classifies known (mostly tracker) domains into one or more categories. The results shown in Table 3 reveal that the most common categories are associated with tracking, advertising, marketing, and analytics. Notably, the most prevalent category among them is Ad Motivated Tracking.

Next, we turn to specific third-party tracker scripts that call `getHighEntropyValues` on most websites. We found that five scripts that retrieve high-entropy hints on most sites come from Google, and most of them are advertising-related. Specifically, googletagmanager.com, googlesyndication.com, and doubleclick.net emerged as the prominent tracker domains among the identified third-party domains as shown in Table 4.

We conducted an analysis to determine which client hints are more frequently retrieved via the `getHighEntropyValues` method by third-party scripts. For this analysis, we used the arguments of `getHighEntropyValues` captured through our JavaScript instrumentation (§3.2). Analyzing function call arguments, we found that the most frequently requested UA client hints via the JavaScript API are `model` and `platformVersion` as shown in Table 5. Notably, a third-party mistakenly calls `getHighEntropyValues` with an mistyped argument, `uaFulVersion`, on seven distinct sites. Additionally, on 174 sites the method is called with the argument *None* by scripts served from the ampproject.org domain. In this case, Chrome only returns low-entropy hints.

Script Category	Num. Sites
Ad Motivated Tracking	44,084
Advertising	43,976
Audience Measurement	40,901
Third-Party Analytics Marketing	40,491
Analytics	40,347
Action Pixels	13,224
Embedded Content	4,523
CDN	4,342
Social - Share	2,338
Ad Fraud	1,339

Table 3: Most common categories of third-party scripts calling `getHighEntropyValues` method. The right-hand column shows the number of distinct websites where we detected at least one call to `getHighEntropyValues` from the respective category. The categorization is based on DuckDuckGo's Tracker Radar dataset [61], where a script is assigned one or more categories based on its domain.

High Entropy API calls		High Entropy API exfiltrations	
Tracker domain	Num. Sites	Tracker domain	Num. Sites
googletagmanager.com	28,929	google-analytics.com	22,517
googlesyndication.com	6,843	google.com	9,325
doubleclick.net	3,633	doubleclick.net	8,853
googletagservices.com	1,414	googlesyndication.com	2,018
googleadservices.com	673	crwdcntrl.net	985
quantserve.com	437	sharethis.com	531
taboola.com	330	gemius.pl	356
clarity.ms	192	taboola.com	315
statcounter.com	161	id5-sync.com	253
wpadmngr.com	152	tynt.com	202

Table 4: Top tracker domains that call getHighEntropyValues (left) and exfiltrate (right) high-entropy values.

Upon analyzing the HTTP URL, POST data, and referrer fields (§3.4), we found that the `fullVersion` or `platformVersion`, along with their corresponding encodings, were being exfiltrated to a third-party domain across 47,691 distinct websites[7]. Notably, in the majority of cases (88.8%), these values were transmitted without any encoding or obfuscation. However, in certain instances, they were encoded using URL encoding (6.8%) or Base64 encoding (4.1%). The righthand column of Table 4 shows the most common domains that high-entropy UA-CHs are exfiltrated to. Google domains including Google Analytics is followed by Lotame (crwdcntrl.net), which offers its audience data to advertisers, marketers and publishers [31]. Our analysis further revealed that on 91.6% of the sites where the *getHighEntropyValues* was called by a tracker script, the high-entropy values were exfiltrated to tracker domains, as shown in

[7]Note that we excluded all UA-CH headers from this analysis.

UA Client Hint	Num. Sites	Num. of Script Domains
model	52,270	970
platformVersion	52,214	1,052
platform	51,529	746
fullVersionList	51,321	792
architecture	51,229	618
bitness	50,874	554
uaFullVersion (deprecated)	50,743	359
wow64	50,132	80
mobile	7,615	283
brands	5,208	462
None	174	1
uaFulVersion	7	1

Table 5: `getHighEntropyValues` arguments sorted by the number of distinct sites they were observed on. The rightmost column indicates the distinct script domains that used the particular hint.

Ent.	UA-CH Header	All	Third Party	Tracking Related
high	Sec-CH-UA-Platform-Version	886	331	134
	Sec-CH-UA-Model	886	329	132
	Sec-CH-UA-Full-Version-List	696	261	67
	Sec-CH-UA-Arch	667	257	63
	Sec-CH-UA-Full-Version	581	217	25
	Sec-CH-UA-Bitness	491	217	25
	Sec-CH-UA-Wow64	401	210	21
low	Sec-CH-UA	89,141	78,476	67,560
	Sec-CH-UA-Mobile	89,141	78,476	67,560
	Sec-CH-UA-Platform	89,141	78,478	67,560

Table 6: The number of distinct sites where CH headers were observed on requests made to all parties, third parties, and tracker domains.

Table 2. Furthermore, despite intercepting WebSocket requests, no exfiltration was observed through this channel.

Our analysis of whether `getHighEntropyValues` is more likely to be called on popular websites yielded a negative result. As Table 10 in Appendix E shows, the Tranco rank of a website does not seem to correlate with more or less access to high-entropy UA-CHs via the JavaScript API.

To gain insights into the motivations behind tracker domains exfiltrating high-entropy values, we conducted a comparative analysis of high-entropy value exfiltration across various categories of third-party domains that *receive* these values. In Appendix F, Table 11 shows that Advertising and Audience Measurement are the top two categories significantly associated with the exfiltration of high-entropy values. Interestingly, these categories align with the categories of the scripts calling the `getHighEntropyValues` method, as shown in Table 3. In fact, the top five categories listed in both tables are exactly the same.

In addition to analyzing scripts involved in fingerprinting, we investigated those accessing high entropy values. Among the 2,190 distinct third-party fingerprinting scripts we identified, a surprisingly low 12.5% make calls to the `getHighEntropyValues` method (Table 12 in Appendix G).

4.2 The collection of User-Agent Client Hint HTTP headers

Recall that UA-CHs are categorized as low and high-entropy depending on how identifying they are (Table 1). Low-entropy CHs are automatically included in every request, while high-entropy CHs require explicit opt-in for first parties or delegation for third parties. In the following tables, low and high entropy hints are presented separately due to this distinction.

Table 6 displays the count of distinct websites where non-empty client hints (CHs) were detected in requests made to both first-party and third-party domains. Notably, `Sec-CH-UA-Platform-Version` and `Sec-CH-Model` were the most frequently observed CHs, mirroring the most received CH via the `getHighEntropyValues` method.

Specifically, `Sec-CH-UA-Platform-Version` was sent on 886 (1%) sites, and among those, it was transmitted to a third-party domain on 331 sites. Our findings were in line with Chrome usage metrics, where `Sec-CH-UA-Platform-Version` was sent on 1.67% of the sites [38]. Further, we found that on 622 sites, the low entropy UA-CHs were not sent in any request. Additional analysis of these cases revealed two main reasons. Firstly, certain sites utilized insecure HTTP connections, leading to the browser not sending UA-CHs. Secondly, a limited number of sites only triggered the initial request, with no subsequent requests made.

As illustrated in Table 13 in Appendix H, the Advertising category stands out as the most common category of third-party domains where high-entropy UA-CH headers were sent to. Similar to the dominant categories associated with `getHighEntropyValues` calls, Advertising and Ad-Motivated tracking categories appear high in this table.

The top two client hints (`Sec-CH-UA-Model` and `Sec-CH-UA-Platform-Version`) were opted-in by first-party domains using the `Accept-CH` header, consistent with the client hints retrieved via the JS method. As depicted in Table 7, our analysis found that `Sec-CH-UA-Model` and `Sec-CH-UA-Platform-Version` were requested in the `Accept-CH` header on 1,046 and 870 distinct sites, respectively. During our analysis, we discovered that certain `Accept-CH` CH header values include typos. Specifically, some of these values lacked the 'Sec-' prefix, such as `ua-platform-version`. For example, `Accept-CH` headers sent by bing.com contained values such as `UA-Full-Version`, `UA-Platform`, and `UA-Arch` (all three incorrect), `Sec-CH-UA-Full-Version-List` (correct). Similarly, issuu.com sent CHs in the `Accept-CH` header without the `Sec` prefix, such as `UA-Arch` and `UA-Full-Version`. In those particular cases, Chrome sent the correctly typed `Sec-CH-UA-Full-Version-List` to bing.com, but did not send any high-entropy hints to issuu.com.

Table 8 shows that high-entropy CH headers are delegated to third-party domains on very few sites (0.4%). Primarily, sites delegate high-entropy UA-CHs to all domains using the '*' character, granting access to the delegated UA CH headers to all nested browsing contexts within the document, regardless of their origin.. fandom.com and google.com are the two domains that were most commonly delegated via the `Permissions`

Ent.	UA-CH Header	Num. Sites
high	Sec-CH-UA-Model	1,046
	Sec-CH-UA-Platform-Version	870
	Sec-CH-UA-Full-Version-List	824
	Sec-CH-UA-Arch	799
	Sec-CH-UA-Full-Version	538
	Sec-CH-UA-Bitness	443
	Sec-CH-UA-Wow64	354
low	Sec-CH-UA-Platform	818
	Sec-CH-UA	434
	Sec-CH-UA-Mobile	403

Table 7: The number of distinct sites where UA-CHs were opted-in via the `Accept-CH` header.

`Policy` header. Only on a few websites high-entropy UA-CHs delegated to specific origins. For example, on chase.com, `Sec-CH-UA-Full-Version-List` is delegated to the subdomains of Chase by specifying "https://*.chase.com". Similarly, certain sites such as idiva.com, indiatime.com, and mensxp.com permit sending high-entropy CHs to a specific origin (ase.clmbtech.com).

Ent.	UA-CH Header	Allow All(*)	Total
high	ch-ua-platform-version	312	338
	ch-ua-model	310	337
	ch-ua-full-version-list	259	266
	ch-ua-arch	261	266
	ch-ua-bitness	221	225
	ch-ua-full-version	225	225
	ch-ua-wow64	218	222
low	ch-ua-platform	218	225
	ch-ua	4	6
	ch-ua-mobile	4	6

Table 8: The number of distinct sites delegating high-entropy UA-CHs via the `Permissions Policy` header. Most websites use '*' instead of specific origins to delegate to all possible origins. These cases are indicated in the 'Allow All' column.

4.3 User-Agent Client Hint opt-in and delegation via HTML

Besides `Permissions Policy` and `Accept-CH` headers, opt-in and delegation of high-entropy UA-CHs can also be accomplished through the use of HTML `<meta>` element attributes and iframes' allow attributes. However, we observed that the usage of these methods for delegation was relatively low. Specifically, `http-equiv='accept-ch'` `<meta>` tag was employed for opt-in by first parties on 117 sites; the iframes' allow attribute was used for delegation to third parties on 32 sites, and the `http-equiv='delegate-ch'` `<meta>` tag was used for delegation to third parties on just 11 sites, as shown in Table 9. A potential reason for the limited use of the HTML-based method is that it introduces a delay in sending the high-entropy client hints since the browser needs to parse the HTML content to discover the opt-in. This may cause the browser to not send the requested high-entropy UA-CHs in the several requests that immediately follow the response—a phenomenon which we observed in our data, but not quantified due to limited scope.

Delegation	Num. Sites
http-equiv='accept-ch'	117
iframe-allow	32
http-equiv='delegate-ch'	11

Table 9: The number of distinct sites where delegation is accomplished through HTML.

4.4 Reduction in high-entropy User-Agent exposure

Finally, we compare the domains that could access high-entropy UA-CHs via the `getHighEntropyValues` method and those that actually use that privilege. Figure 5 shows that 25,052 (61%) of the 41,229 distinct third-party domains encountered in our crawl serve active content with resource type reported by Puppeteer as "script" or "document" — the latter of which can contain inline scripts. Scripts from these domains *can* start collecting the high-entropy UA-CHs overnight, by adding a call to the `getHighEntropyValues` method. This could triple the number of distinct domains that access such high-entropy UA-CHs, which currently amount to 9,095 (22%) distinct third-party domains. While high-entropy UA-CHs can be accessed by third parties via HTTP headers, our results suggest that third-party delegation is rarely observed.(Table 8).

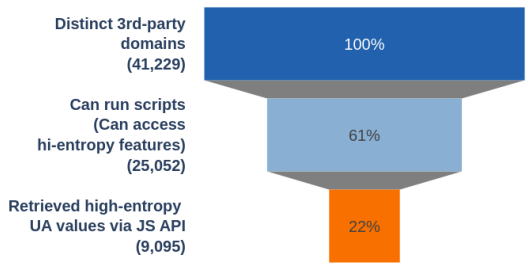

Figure 5: Comparing distinct 3rd-party domains that *can* run scripts and thus *can* access high-entropy UA-CHs and those that *accessed* high-entropy UA-CHs via the JavaScript API.

5 DISCUSSION

The UA reduction efforts seem to have achieved their goal of limiting the potentially identifying information readily available in the UA HTTP header. Thanks to the *off-by-default* design of high-entropy UA-CH HTTP headers, we found that less than 1% of third

parties receive high-entropy client hints over HTTP. On the other hand, the same high-entropy client hints are made accessible to scripts without any check for their origin or security context. This unfettered access enabled the collection and exfiltration of high-entropy UA-CHs by tracker scripts on close to 60% of the websites we studied. Particularly notable is that Google, primarily through its ads and analytics-related scripts, stands out as the most prevalent user of the `getHighEntropyValues` method and owns the most common domain (google-analytics.com) to which high-entropy UA-CHs are exfiltrated to by scripts (Table 4). We also note that while UA reduction efforts aim to limit covert tracking, Google's recently introduced Server-Side Tagging (SST) mechanism has almost the opposite effect [53]. SST shifts data sharing with third parties from client to server-side, potentially hiding the third parties that collect web users' detailed online activities.

Somehow surprisingly, even if Chrome engineers decide to further reduce identifying information available through the UA-CHs, they may not be allowed to do so—at least in the short term. Responding to the UK Competition and Markets Authority's investigation into the Privacy Sandbox project, Google has committed to ensuring that all the information previously present in the user-agent string will remain accessible through the UA CHs until the third-party cookie deprecation [35]. This was part of the commitments made to limit the potential anti-competition effects of the Privacy Sandbox project.

Finally, one might also question the relative significance of privacy risks posed by passive fingerprinting in relation to the widespread use of active and overt tracking—where Google is the most prevalent tracker entity through its ads, analytics and tag manager products [86]. Further, it is unclear to us whether overt cross-site tracking methods such as (active) browser fingerprinting or cookie syncing are more transparent or auditable from an end-user's perspective. While we recognize the improved transparency in our research, web measurement studies like ours cannot offer privacy assurances akin to a private-by-design system.

5.1 Limitations

Similar to other web measurement research, our study has several limitations regarding representativeness, reproducibility [63] and coverage. With a single form factor (desktop) and vantage point in the US, we cannot claim our results are representative of real users' experiences around the world. Further, websites may detect our crawler as an automated bot, and treat it differently [104]. We rely on TRC's anti-bot measures [58] which thwarts bot detection to a certain extent, but we acknowledge that its efficacy may be limited. With respect to coverage, our crawler only visited home pages and could not visit pages behind paywalls or login walls.

A specific limitation of our measurement was the inability to capture opt-ins at the connection time via HTTP/2 and HTTP/3 frames [22, 45] since those features are disabled in Puppeteer due to incompatibility with the Chrome DevTools Protocol [14]. We believe this limitation did not affect our prevalence measurements, but could skew the measurement of *mechanisms* used to convey the UA-CH opt-ins. In particular, our results do not capture the use of HTTP/2 or HTTP/3 frames for opting-in to high-entropy UA-CHs. In order to verify that we have not missed any HTTP/2 or HTTP/3 `Accept_CH` frames, we made sure that we can attribute all high-entropy UA-CH headers to a corresponding HTTP `Accept-CH`, `Permissions Policy` headers or `<meta>` tags, except on five websites. The most likely reason why we cannot pin down the opt-in mechanism on those five websites is `<meta>` tags that are temporarily injected and removed before we query the DOM, which is another minor limitation of our approach. Regardless, we note that these limitations do not affect our core measurement of high-entropy JavaScript API usage, which does not require any opt-in or delegation.

Finally, our measurement presents a snapshot of the current state of the web, which may change over time. For instance, while we found `getHighEntropyValues` method to be called on 59.2% of the websites, the older, now-reduced alternative `navigator.userAgent` property was accessed on 88.3% of the websites. We expect the number of websites where `getHighEntropyValues` is called to increase over time, as also shown by the upwards trend in the Chrome telemetry data for this particular JavaScript method [37].

6 CONCLUSION

We presented the first empirical study of the impact of user-agent string reduction and the introduction of high-entropy user-agent client hints in the Chrome browser. Through an extensive analysis of the top 100K websites using an instrumented crawler, we quantified the collection and exfiltration of high-entropy browser features via UA-CH HTTP headers and the newly introduced JavaScript API. Our findings showed that third-party scripts accessed high-entropy user agent client hints on nearly 60% of the analyzed sites. These scripts consist primarily of tracking and advertising scripts, with a notable presence of scripts owned by Google. Over 90% of the websites where high-entropy client hints were accessed via the JavaScript API, the obtained hints were exfiltrated to remote servers by tracker scripts. We find the use of high-entropy UA-CH headers to be very limited: only 1.3% of the first parties opt-in to receive high-entropy UA-CH headers, and even fewer websites (0.4%) delegate high-entropy hints to third-party domains. Overall, the efforts to reduce UA string identification appeared to be effective in minimizing the number of third parties that received the full UA string in HTTP headers, with only 3% of third-party domains receiving high-entropy UA-CH headers. Our study suggests that users and website operators may prefer JavaScript access for managing high-entropy browser features due to performance and operational considerations. In conclusion, while the UA reduction efforts appear to be effective for curbing passive browser fingerprinting, the introduction of high-entropy user-agent client hints has limited the overall privacy benefits of this major change. Given the extensive collection and exfiltration of highly identifying client hints by tracker scripts, we believe browser vendors should consider imposing stricter controls on scripted access to high-entropy user-agent client hints.

ACKNOWLEDGMENTS

We thank Martin Thomson, Iness Ben Guirat, Zahra Moti and Luqman Zagi for their comments. Asuman Senol received funding from CYD Campus, armasuisse Science and Technology.

REFERENCES

[1] 1994-05-03. *Request Headers in the HTTP protocol.* https://www.w3.org/Protocols/HTTP/HTRQ_Headers.html#user-agent [Online; accessed 22. Jul. 2023].

[2] 2008. Emotional Branding - Hispanic Marketing & Public Relations website and podcast. https://hispanicmpr.com/resources/hmpr-products/emotional-branding [Online; accessed 28. Jul. 2023].

[3] 2008-09-03. *WebAIM: History of the browser user-agent string.* https://webaim.org/blog/user-agent-string-history [Online; accessed 22. Jul. 2023].

[4] 2013-11-05. *935216 - (client-hints) Implement Client-Hints HTTP header.* https://bugzilla.mozilla.org/show_bug.cgi?id=935216 [Online; accessed 19. Jul. 2023].

[5] 2017. *180365 – Limit user agent versioning to an upper bound.* https://bugs.webkit.org/show_bug.cgi?id=180365 [Online; accessed 30. Jul. 2023].

[6] 2018-02-22. *Safari 11.1.* https://developer.apple.com/library/archive/releasenotes/General/WhatsNewInSafari/Articles/Safari_11_1.html#//apple_ref/doc/uid/TP40014305-CH14-SW7 [Online; accessed 09. Jul. 2023].

[7] 2019. *Mitigating Browser Fingerprinting in Web Specifications.* https://www.w3.org/TR/fingerprinting-guidance [Online; accessed 28. Jul. 2023].

[8] 2020-01-14. *1609304 - Reduce Gecko's User-Agent strings.* https://bugzilla.mozilla.org/show_bug.cgi?id=1609304 [Online; accessed 19. Jul. 2023].

[9] 2021-05-24. *Client hints not sent in headless chrome.* https://bugs.chromium.org/p/chromium/issues/detail?id=1212793 [Online; accessed 01. Jul. 2023].

[10] 2021-10-14. Intent to Prototype: User-Agent Client Hints GREASE Update. https://groups.google.com/a/chromium.org/g/blink-dev/c/ueudFsZzT1M [Online; accessed 22. Jul. 2023].

[11] 2021-11-02. *Contribute to WICG/ua-client-hints.* https://github.com/WICG/ua-client-hints/issues?q=is%3Aissue+author%3Aothermaciej [Online; accessed 01. Jul. 2023].

[12] 2022. *Solutions: Syndicated Audiences: CultureCode® - Claritas LLC.* https://claritas.com/culture-code [Online; accessed 28. Jul. 2023].

[13] 2022-01-13. *Implement Navigator.userAgentData.* https://bugzilla.mozilla.org/show_bug.cgi?id=1750143 [Online; accessed 14. Jul. 2023].

[14] 2023. *1348106 - AcceptCHFrame breaks expected order of CDP events - chromium.* https://crbug.com/1348106 [Online; accessed 1. Aug. 2023].

[15] 2023. *182629 – [macOS, iOS] Expose OS marketing version in UserAgent.* https://bugs.webkit.org/show_bug.cgi?id=182629#c6 [Online; accessed 28. Jul. 2023].

[16] 2023. *About Events API | TikTok For Business .* https://ads.tiktok.com/help/article/events-api [Online; accessed 28. Jul. 2023].

[17] 2023. *Advertising Strategies for Targeting U.S. Hispanics.* https://www.comscore.com/lat/Prensa-y-Eventos/Blog/Advertising-Strategies-for-Targeting-U.S.-Hispanics [Online; accessed 28. Jul. 2023].

[18] 2023. *Analysis of Google's Privacy Budget Proposal | The Mozilla Blog.* https://blog.mozilla.org/en/mozilla/google-privacy-budget-analysis [Online; accessed 30. Jul. 2023].

[19] 2023. *Bulletin: Chrome's User-Agent Client Hints.* https://support.appsflyer.com/hc/en-us/articles/12305445230737-Bulletin-Chrome-s-User-Agent-Client-Hints [Online; accessed 28. Jul. 2023].

[20] 2023. *Changing the UA String - MozillaWiki.* https://wiki.mozilla.org/Changing_the_UA_String [Online; accessed 17. Jul. 2023].

[21] 2023. *Client Hint Reliability.* https://datatracker.ietf.org/doc/html/draft-davidben-http-client-hint-reliability#section-4 [Online; accessed 28. Jul. 2023].

[22] 2023. *client-hints-infrastructure/reliability.md at main · WICG/client-hints-infrastructure.* https://github.com/WICG/client-hints-infrastructure/blob/main/reliability.md#connection-level-settings [Online; accessed 28. Jul. 2023].

[23] 2023. *Early browser API accesses and function calls are missed.* https://github.com/duckduckgo/tracker-radar-collector/issues/77 [Online; accessed 29. Jul. 2023].

[24] 2023. *EasyList.* https://easylist.to/easylist/easylist.txt [Online; accessed 10. Jul. 2023].

[25] 2023. *EasyPrivacy.* https://easylist.to/easylist/easyprivacy.txt [Online; accessed 10. Jul. 2023].

[26] 2023. *Extend Sec-CH-UA to include a boolean field to identify if browser requires an update · Issue #76 · WICG/ua-client-hints.* https://github.com/WICG/ua-client-hints/issues/76 [Online; accessed 29. Jul. 2023].

[27] 2023. *@gorhill/ubo-core.* https://www.npmjs.com/package/@gorhill/ubo-core [Online; accessed 10. Jul. 2023].

[28] 2023. *How To Target Facebook Ads To "Immigration to the United States" Audience | AdTargeting.* https://adtargeting.io/facebook-ad-targeting/immigration-to-the-united-states [Online; accessed 28. Jul. 2023].

[29] 2023. *HTTP Client hints - HTTP | MDN.* https://developer.mozilla.org/en-US/docs/Web/HTTP/Client_hints [Online; accessed 31. Jul. 2023].

[30] 2023. isbot. https://github.com/omrilotan/isbot [Online; accessed 30. Jul. 2023].

[31] 2023. Lotame. https://better.fyi/trackers/crwdcntrl.net/ [Online; accessed 31. Jul. 2023].

[32] 2023. *Merge pull request #253 from kyraseevers/issue248 · WICG/ua-client-hints@56df868.* https://github.com/WICG/ua-client-hints/commit/56df868c6c3564461b4c0a42ea603b80fa94c8d2 [Online; accessed 28. Jul. 2023].

[33] 2023. *Mozilla Specification Positions.* https://mozilla.github.io/standards-positions/#ua-client-hints [Online; accessed 23. Jun. 2023].

[34] 2023. *NavigatorUAData - Web APIs | MDN.* https://developer.mozilla.org/en-US/docs/Web/API/NavigatorUAData [Online; accessed 22. Jul. 2023].

[35] 2023. *Notice of intention to accept modified commitments offered by Google in relation to its Privacy Sandbox Proposals.* https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1036204/211126_FINAL_modification_notice.pdf [Online; accessed 8. Aug. 2023].

[36] 2023. *Page.setUserAgent() method.* https://pptr.dev/api/puppeteer.page.setuseragent [Online; accessed 20. Jul. 2023].

[37] 2023. *Percentage of page loads over time - GetHighEntropyValues .* https://chromestatus.com/metrics/feature/timeline/popularity/4520 [Online; accessed 25. Jul. 2023].

[38] 2023. *Percentage of page loads over time - PlatformVersion.* https://chromestatus.com/metrics/feature/timeline/popularity/3273 [Online; accessed 31. Jul. 2023].

[39] 2023. *Prepare for phasing out third-party cookies - Chrome Developers.* https://developer.chrome.com/en/docs/privacy-sandbox/third-party-cookie-phase-out [Online; accessed 28. Jul. 2023].

[40] 2023. *privacy-budget.* https://github.com/mikewest/privacy-budget [Online; accessed 30. Jul. 2023].

[41] 2023. *Protecting Your Privacy Online.* https://privacysandbox.com/ [Online; accessed 13. Jul. 2023].

[42] 2023. *Qualys SSL Labs - Projects / HTTP Client Fingerprinting Using SSL Handshake Analysis.* https://www.ssllabs.com/projects/client-fingerprinting [Online; accessed 30. Jul. 2023].

[43] 2023. *Sec-CH-Bot for self declared 'good bots' that want to avoid analytics pollution · Issue #119 · WICG/ua-client-hints.* https://github.com/WICG/ua-client-hints/issues/119 [Online; accessed 29. Jul. 2023].

[44] 2023. *The UK Uses Targeted Facebook Ads To Deter Migrants. Now Meta Is Releasing the Data.* https://newlinesmag.com/reportage/the-uk-uses-targeted-facebook-ads-to-deter-migrants-now-meta-is-releasing-the-data [Online; accessed 28. Jul. 2023].

[45] 2023. *TLS Application-Layer Protocol Settings Extension.* https://datatracker.ietf.org/doc/html/draft-vvv-tls-alps [Online; accessed 28. Jul. 2023].

[46] 2023. tls-fingerprinting/fingerprintls at master · LeeBrotherston/tls-fingerprinting. https://github.com/LeeBrotherston/tls-fingerprinting/tree/master/fingerprintls [Online; accessed 30. Jul. 2023].

[47] 2023. *Types of documents W3C publishes.* https://www.w3.org/standards/types/#x2-pre-standardization-proposals-notes [Online; accessed 31. Jul. 2023].

[48] 2023. *uBlock Origin - An efficient blocker for Chromium and Firefox. Fast and lean.* https://github.com/gorhill/uBlock [Online; accessed 10. Jul. 2023].

[49] 2023. *User-Agent Client Hints API.* https://developer.mozilla.org/en-US/docs/Web/API/User-Agent_Client_Hints_API [Online; accessed 28. Jul. 2023].

[50] 2023. *User-Agent Reduction.* https://www.chromium.org/updates/ua-reduction [Online; accessed 13. Jul. 2023].

[51] 2023. *User-Agent reduction - Chrome Developers.* https://developer.chrome.com/docs/privacy-sandbox/user-agent [Online; accessed 28. Jul. 2023].

[52] 2023. *User-Agent targeting.* https://developers.google.com/authorized-buyers/rtb/useragent-targeting [Online; accessed 28. Jul. 2023].

[53] 2023. *Why and when to use server-side tagging?* https://developers.google.com/tag-platform/learn/sst-fundamentals/3-why-and-when-sst [Online; accessed 8. Aug. 2023].

[54] 2023-04-16. *Sec-CH-UA Brand being abused by sites, blocking clients.* https://bugs.chromium.org/p/chromium/issues/detail?id=1433548 [Online; accessed 14. Jul. 2023].

[55] Victor Tan Ali Beyad. 2022-02-24. *User-Agent Reduction deprecation trial.* https://developer.chrome.com/en/blog/user-agent-reduction-deprecation-trial/ [Online; accessed 09. Jul. 2023].

[56] Blake Anderson and David McGrew. 2017. OS fingerprinting: New techniques and a study of information gain and obfuscation. In *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–9.

[57] D. Benjamin. 2020-01-01. *Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility.* https://datatracker.ietf.org/doc/html/rfc8701 [Online; accessed 12. Jul. 2023].

[58] Konrad Dzwinel et al. Brad Slayter, Sam Macbeth. 2021. *DuckDuckGo Tracker Radar Collector.* https://github.com/duckduckgo/tracker-radar-collector [Online; accessed 01. Jan. 2023].

[59] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-) browser fingerprinting via OS and hardware level features. In *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society.

[60] Contributors to Wikimedia projects. 2023. *WebKit - Wikipedia.* https://en.wikipedia.org/w/index.php?title=WebKit&oldid=1164531541 [Online; accessed 1. Aug. 2023].

[61] Konrad Dzwinel et al. David Harbage, Sam Macbeth. 2020. *DuckDuckGo Tracker Radar.* https://github.com/duckduckgo/tracker-radar [Online; accessed 01. Jan. 2023].

[62] Jon Davis. 2017-12-20. *Safari 11.1.* https://webkit.org/blog/8042/release-notes-for-safari-technology-preview-46/ [Online; accessed 09. Jul. 2023].

[63] Nurullah Demir, Matteo Große-Kampmann, Tobias Urban, Christian Wressnegger, Thorsten Holz, and Norbert Pohlmann. 2022. Reproducibility and replicability of web measurement studies. In *Proceedings of the ACM Web Conference 2022*. 533–544.

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

[65] Peter Eckersley. 2010. How unique is your web browser? *Privacy Enhancing Technologies* (2010), 1–18. https://doi.org/10.1007/978-3-642-14527-8_1

[66] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. 2018. I never signed up for this! Privacy implications of email tracking. *Proc. Priv. Enhancing Technol.* 2018, 1 (2018), 109–126.

[67] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 1388–1401.

[68] Martin Grill and Martin Rehák. 2014. Malware detection using HTTP user-agent discrepancy identification. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 221–226.

[69] We Are All Human. 2023. *New Study Reveals Hispanics Feel More Excluded From The American Culture And Disregarded By Big Brands.* https://www.prnewswire.com/news-releases/new-study-reveals-hispanics-feel-more-excluded-from-the-american-culture-and-disregarded-by-big-brands-301857672.html

[70] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2021. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1143–1161.

[71] Nikhil Jha, Martino Trevisan, Luca Vassio, and Marco Mellia. 2022. The Internet with privacy policies: Measuring the Web upon consent. *ACM Transactions on the Web (TWEB)* 16, 3 (2022), 1–24.

[72] Barry Pollard Johannes Henkel. 2021-03-09. *Adding Rank Magnitude to the CrUX Report in BigQuery.* https://developer.chrome.com/blog/crux-rank-magnitude/ [Online; accessed 13. Jul. 2023].

[73] Seb Joseph. 2021. *'The elephant in the room': Companies persist with fingerprinting as a workaround to Apple's new privacy rules.* https://digiday.com/media/the-elephant-in-the-room-companies-persist-with-fingerprinting-as-a-workaround-to-apples-new-privacy-rules

[74] Terry Parris Julia Angwin, Jr. 2023. *Facebook Lets Advertisers Exclude Users by Race.* https://www.propublica.org/article/facebook-lets-advertisers-exclude-users-by-race [Online; accessed 28. Jul. 2023].

[75] Jeff Kline, Paul Barford, Aaron Cahn, and Joel Sommers. 2017. On the structure and characteristics of user agent string. In *Proceedings of the 2017 Internet Measurement Conference*. 184–190.

[76] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. 2005. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing* 2, 2 (2005), 93–108.

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

[78] Kevin K. Lee. 2022-04-20. *Controlling browser features with Permissions Policy.* https://developer.chrome.com/docs/privacy-sandbox/permissions-policy/ [Online; accessed 09. Jul. 2023].

[79] Marek. 2012. *SSL fingerprinting for p0f — Idea of the day.* https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f [Online; accessed 30. Jul. 2023].

[80] nikhiljha95 Martino Trevisan, Antonino Musmeci. 2022-04-13. *Priv-Accept.* https://github.com/marty90/priv-accept [Online; accessed 13. Jul. 2023].

[81] Jonathan R Mayer and John C Mitchell. 2012. Third-party web tracking: Policy and technology. In *2012 IEEE symposium on security and privacy*. IEEE, 413–427.

[82] Yoav Weiss Mike Taylor. 2023. *User Agent Hints.* https://wicg.github.io/ua-client-hints [Online; accessed 14. Jul. 2023].

[83] Kyle Morehouse. 2022. Increase Ad ROI With Audience Suppression. https://blog.adobe.com/en/publish/2017/01/24/increase-ad-roi-audience-suppression [Online; accessed 28. Jul. 2023].

[84] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* 2012 (2012).

[85] Robert Nyman. 2013-09-12. *User-Agent detection, history and checklist – Mozilla Hacks - the Web developer blog.* https://hacks.mozilla.org/2013/09/user-agent-detection-history-and-checklist [Online; accessed 14. Jul. 2023].

[86] Barry Pollard. 2023. *Third Parties | 2021 | The Web Almanac by HTTP Archive.* https://almanac.httparchive.org/en/2021/third-parties

[87] Ali Rasaii, Shivani Singh, Devashish Gosain, and Oliver Gasser. 2023. Exploring the Cookieverse: A Multi-Perspective Analysis of Web Cookies. In *International Conference on Passive and Active Network Measurement*. Springer, 623–651.

[88] Yoav Weiss Rowan Merewood. 2021-09-10. *User-Agent Client-Hints.* https://developer.chrome.com/en/articles/user-agent-client-hints/ [Online; accessed 09. Jul. 2023].

[89] Dennis Schubert. 2022-12-15. *Freeze 'rv:' segment in the User Agent string to 'rv:109.0' to avoid erroneous IE11 detection.* https://bugzilla.mozilla.org/show_bug.cgi?id=1805967 [Online; accessed 09. Jul. 2023].

[90] Henri Sivonen. 2021-07-12. *Downgrade User Agent Client Hints to 'harmful' · Issue #552 · mozilla/standards-positions.* https://github.com/mozilla/standards-positions/issues/552 [Online; accessed 19. Jul. 2023].

[91] Maciej Stachowiak. 2021-11-02. *getHighEntropyValues should not return all values, only values that have been opted in.* https://github.com/WICG/ua-client-hints/issues/151 [Online; accessed 23. Jun. 2023].

[92] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. 2016. Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms. *Proc. Priv. Enhancing Technol.* 2016, 1 (2016), 20–33.

[93] Yousouf Taghzouti, Antoine Zimmermann, and Maxime Lefrançois. 2022. Content negotiation on the Web: State of the art. *arXiv preprint arXiv:2204.10097* (2022).

[94] Mike Taylor. 2021-05-27. *User-Agent Client Hints & UA Reduction.* https://github.com/w3ctag/design-reviews/issues/640 [Online; accessed 23. Jun. 2023].

[95] Mike Taylor. 2021-05-28. *The hidden meaning of 537.36 in the Chromium User-Agent string.* https://miketaylr.com/posts/2021/05/webkit-537-36-meaning.html [Online; accessed 22. Jul. 2023].

[96] Mike Taylor. 2022-11-04. *Return empty string on Linux for platform-version hint.* https://bugs.chromium.org/p/chromium/issues/detail?id=1381304&q=platformVersion&can=2 [Online; accessed 02. Jul. 2023].

[97] Martin Thomson. 2021-09-29. *UA Client Hints is less than harmful.* https://github.com/mozilla/standards-positions/pull/579 [Online; accessed 02. Jul. 2023].

[98] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 728–741.

[99] Yoav Weiss. 2020-01-14. *Intent to deprecate and freeze: The user-agent string.* https://groups.google.com/a/chromium.org/g/blink-dev/c/-2JIRNMWJ7s/m/yHe4tQNLCgAJ [Online; accessed 13. Jul. 2023].

[100] Alexandra White. 2021-11-09. *User-Agent reduction.* https://developer.chrome.com/docs/privacy-sandbox/user-agent/ [Online; accessed 8. Aug. 2023].

[101] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. 2012. Host Fingerprinting and Tracking on the Web: Privacy and Security Implications.. In *NDSS*, Vol. 62. 66.

[102] YngveNPettersen. 2022-02-16. *Brand info is being used to block clients.* https://github.com/WICG/ua-client-hints/issues/293 [Online; accessed 14. Jul. 2023].

[103] Michal Zalewski. 2019-02-21. *p0f unofficial git repo.* https://github.com/p0f/p0f [Online; accessed 01. Jul. 2023].

[104] David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén, and Martin Lopatka. 2020. The representativeness of automated web crawls as a surrogate for human browsing. In *Proceedings of The Web Conference 2020*. 167–178.

A NAVIGATOR.USERAGENTDATA INTERFACE DESCRIPTION (IDL) [82]

```
navigator.userAgentData
// OUTPUT:
"brands":[
    {
        "brand":" Not A;Brand",
        "version":"24"
    },
    {
        "brand":"Chromium",
        "version":"113"
    },
    {
        "brand":"Google Chrome",
        "version":"113"
],
"mobile": false,
"platform": "Linux"
```

Listing 4: Sample output of navigator.userAgent interface's properties

```
dictionary NavigatorUABrandVersion {
  DOMString brand;
  DOMString version;
};
```

```
dictionary UADataValues {
  DOMString architecture;
  DOMString bitness;
  sequence<NavigatorUABrandVersion> brands;
  DOMString formFactor;
  sequence<NavigatorUABrandVersion> fullVersionList;
  DOMString model;
  boolean mobile;
  DOMString platform;
  DOMString platformVersion;
  DOMString uaFullVersion; // deprecated in favor of
     fullVersionList
  boolean wow64;
};

dictionary UALowEntropyJSON {
  sequence<NavigatorUABrandVersion> brands;
  boolean mobile;
  DOMString platform;
};

[Exposed=(Window,Worker)]
interface NavigatorUAData {
  readonly attribute FrozenArray<NavigatorUABrandVersion>
      brands;
  readonly attribute boolean mobile;
  readonly attribute DOMString platform;
  Promise<UADataValues> getHighEntropyValues(sequence<
     DOMString> hints);
  UALowEntropyJSON toJSON();
};

interface mixin NavigatorUA {
  [SecureContext] readonly attribute NavigatorUAData
     userAgentData;
};

Navigator includes NavigatorUA;
WorkerNavigator includes NavigatorUA;
```

Listing 5: NavigatorUAData interface

B CRAWLER'S UA OVERRIDE VALUES

We used the following details to override the crawler's UA string and client hints via the *page.setUserAgent* [36] method of the Chrome DevTools Protocol.

```
brands: [
    {brand: 'Chromium', version: '113'},
    {brand: 'Not-A.Brand', version: '24'}
],
fullVersionList: [
    {brand: 'Chromium', version: '113.0.5672.63'},
    {brand: 'Not-A.Brand', version: '24.0.0.0'}
],
fullVersion: "113.0.5672.63",
platform: 'Linux',
platformVersion: '5.15.0',
architecture: 'x86',
model: '',
mobile: false,
bitness: '64',
wow64: false,
```

Listing 6: Overriden UA Values

C FINGERPRINTING DETECTION HEURISTICS

Below you can find the heuristics used for the detection of fingerprinting attempts in this study. These heuristics were first proposed by Englehardt and Narayanan [67] and then improved by Iqbal et. al [70].

Canvas Fingerprinting Identification: A script categorized as a canvas fingerprinting script based on the following criteria:

(1) The script calls the `fillText` or `strokeText` methods to write text on a canvas element, and style adjustments are made using the `fillStyle` or `strokeStyle` methods of the rendering context.
(2) The script uses the `toDataURL` method to extract the canvas image.
(3) The script does not call the `save`, `restore`, or `addEventListener` methods on the canvas element.

WebRTC Fingerprinting Identification: A script is categorized as a WebRTC fingerprinting script according to the following conditions:

(1) The script calls the `createDataChannel` or `createOffer` methods of the WebRTC peer connection.
(2) The script calls the `onicecandidate` or `localDescription` methods of the WebRTC peer connection.

Canvas Font Fingerprinting Identification: A script is recognized as a canvas font fingerprinting script based on these guidelines:

(1) The script modifies the `font` property of a canvas element to include more than 20 different fonts.
(2) The script calls the `measureText` method of the rendering context more than 20 times.

AudioContext Fingerprinting Identification: A script is identified as an AudioContext fingerprinting script if it satisfies the following conditions:

(1) The script calls any of the following methods of the audio context: `createOscillator`, `createDynamicsCompressor`, `destination`, `startRendering`, or `oncomplete`.

D SUPPORTED ENCODINGS & HASHES FOR LEAK DETECTION

Encodings: Base16, Base32, Base58, Base64, Urlencode, Entity, LZstring, Hashes: MD5, SHA1, SHA256, SHA512

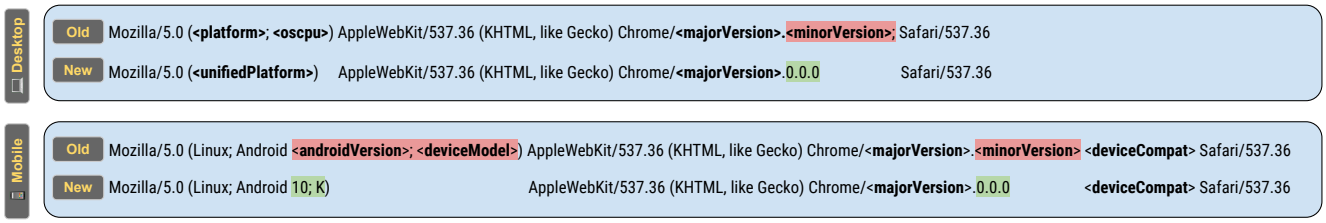

Figure 6: User-agent string format change for desktop and mobile Chrome browsers after UA reduction.

E POPULARITY OF SITES WHICH CALL GETHIGHENTROPYVALUES

	Sites Calling getHighEntropyValues	
Rank Interval	**Website (count)**	**Website (%)**
1 to 1K	508	55.8%
1K to 5K	2,029	57.2%
5K to 10K	2,671	59.6%
10K to 50K	21,092	58.9%
50K to 100K	26,092	58.0%
1 to 100K	52,392	58.4%

Table 10: The distribution of CrUX websites that call getHighEntropyValues. Results are sliced by site rank. and based on successful visits.

F PURPOSES OF THIRD-PARTY SCRIPTS THAT EXFILTRATE HIGH-ENTROPY VALUES

Category	Num. Sites
Advertising	41,124
Audience Measurement	27,526
Ad Motivated Tracking	27,253
Third-Party Analytics Marketing	26,220
Analytics	25,509
CDN	15,512
Online Payment	15,084
Social - Share	2,471
Embedded Content	2,154
Action Pixels	625

Table 11: The top third-party domain categories along with the respective counts of websites where we observed at least one third-party domain exfiltrates high-entropy UA values to their endpoints. The categorization is based on Tracker Radar's domains [61], acknowledging the possibility of a website belonging to multiple categories simultaneously.

G SCRIPTS CALLING GETHIGHENTROPYVALUES AND ALSO FINGERPRINTING

	All	Third party	Tracking related
Fingerprinting	4,689	2,190	784
FPing and `getHighEntropyValues` calls	545	274	157

Table 12: We found 2,190 distinct third-party fingerprinting scripts. 12.5% of them also call the `getHighEntropyValues` method.

H PURPOSES OF THIRD-PARTY SCRIPTS THAT RECEIVE HIGH-ENTROPY UA-CH HEADERS

Category	Num. Sites
Advertising	99
Ad Motivated Tracking	96
Embedded Content	90
CDN	86
Online Payment	65
Analytics	65
Audience Measurement	64
Third-Party Analytics Marketing	53
Action Pixels	22
Ad Fraud	22

Table 13: The top third-party domain categories and the corresponding number of websites where at least one high-entropy UA-CHs was sent to a third party.