

UA-RADAR: Exploring the Impact of User Agents on the Web

Jean Luc Intumwayase
intumwayase@gmail.com

Univ. Lille, Inria, CNRS, UMR CRISTAL 9189
Lille, France

Pierre Laperdrix
pierre.laperdrix@inria.fr

Univ. Lille, Inria, CNRS, UMR CRISTAL 9189
Lille, France

Imane Fouad
imane.fouad@inria.fr

Univ. Lille, Inria, CNRS, UMR CRISTAL 9189
Lille, France

Romain Rouvoy
romain.rouvoy@univ-lille.fr

Univ. Lille, Inria, CNRS, UMR CRISTAL 9189
Lille, France

ABSTRACT

In the early days of the web, giving the same web page to different browsers could provide very different results. As the rendering engine behind each browser would differ, some elements of a page could break or be positioned in the wrong location. At that time, the *User Agent* (UA) string was introduced for content negotiation. By knowing the browser used to connect to the server, a developer could provide a web page that was tailored for that specific browser to remove any usability problems. Over the past three decades, the UA string remained exposed by browsers, but its current usefulness is being debated. Browsers now adopt the exact same standards and use the same languages to display the same content to users, bringing the question if the content of the UA string is still relevant today, or if it is a relic of the past. Moreover, the diversity of means to browse the web has become so large that the UA string is one of the top contributors to tracking users in the field of browser fingerprinting, bringing a sense of urgency to deprecate it.

In this paper, our goal is to understand the impact of the UA on the web and if this legacy string is still actively used to adapt the content served to users. We introduce UA-RADAR, a web page similarity measurement tool that compares in-depth two web pages from the code to their actual rendering, and highlights the similarities it finds. We crawled 270,048 web pages from 11,252 domains using 3 different browsers and 2 different UA strings to observe that 100% of the web pages were similar before any JavaScript was executed, demonstrating the absence of differential serving. Our experiments also show that only a very small number of websites are affected by the lack of UA information, which can be fixed in most cases by updating code to become browser-agnostic. Our study brings some proof that it may be time to turn the page on the UA string and retire it from current web browsers.

KEYWORDS

User Agent; Browsers; Web Page Similarity; Differential Serving; Content Adaptation

1 INTRODUCTION

In the early days of the web, web browsers had different technological stacks and would not interpret HTML tags the exact same way [19]. This created usability problems as the exact same version of a web page would render differently on different browsers. To remedy this problem, each browser started to include a *User Agent* (UA) header that would expose the browser and its version to the

server. Web developers could then provide a version that was tailored to the user's browser so that the website would appear as intended with all the elements in the right place.

In 2023, more than 30 years after it was first officially introduced [50], the UA string is still being used and its history is long, granular, and complex [24]. What was first introduced as a tool to help servers to deliver the most optimized content to users became a source of competition and now tracking [2]. In particular, UA exposed by browsers can be leveraged by browser fingerprinting, which has seen a steady rise in the past decade [28]. By running a little script on a web page, a server can collect a wide range of information on the device being used by the user from the browser and its version to the size of the screen or the GPU. The diversity of today's devices and configurations is so large that it is possible to identify users based only on this information. No other identifiers like cookies are needed to track users on the Internet if a fingerprint is precise enough. Because of the danger posed by fingerprinting, some browser vendors started to make modifications to limit the information revealed by the browser. One such initiative is the UA Client Hints by Google [53], whose goal is to freeze the UA string as it is one of the most revealing information in fingerprints [13, 27].

In this paper, we investigate the impact of the UA string on the web and whether servers still leverage it to adapt the content that is served to users. We introduce UA-RADAR, a web page similarity measurement tool to assess the impact of restricting the User-Agent request-header field, the `navigator.userAgent`, and other identifying information in the `Navigator` object on the web. We crawl 270,048 web pages from 11,252 domains using standard and so-called *none* browsers, and we observe 100% similarity of the web pages before the execution of JavaScript, demonstrating the absence of differential serving. However, 8.4% of the web pages change after the execution of JavaScript, hence highlighting dependency on UA for content adaptation. We conduct a change impact analysis on UA-dependent web pages and find third-party scripts from ads, bot detection, and content delivery network services behind the changes in the web pages.

This paper addresses the following research questions:

- **RQ1:** Do modern websites adapt to the UA?
- **RQ2:** What are the changes created by different UA? What are their causes?
- **RQ3:** What is the impact of removing identifying information from the UA?

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 details our methodology and

how we measure web similarity in the wild. Section 4 goes over details of our crawl and analysis of our dataset. Section 5 discusses the impact of UA on the web, based on our findings. Section 7 discusses the threats to the validity of our work and Section 8 concludes the paper.

2 RELATED WORK

2.1 User Agent on the Web

In the literature, UA has been studied for security improvements, network monitoring, Internet traffic analysis, and user behavior analysis [9, 22–24, 26, 34, 37]. We want to highlight here 2 specific areas that are relevant to this study.

The threat of the UA to privacy. In the past decade, a tracking technique known as browser fingerprinting [28] has grown a lot. By collecting attributes in a web browser from HTTP headers and JavaScript, a script can build the *fingerprint* of a device that can be used later on to identify a user [13]. The diversity of hardware and software configuration is so broad that the combination of all collected attributes can be unique. According to three large-scale fingerprinting studies [13, 17, 27], the HTTP user agent is one of the most revealing attributes in a fingerprint, as it always ended up in the top 3 of collected attributes with the most entropy. In particular, this header on mobile can even reveal the exact model of the user’s smartphone along with the phone carrier which is very worrisome for privacy [27]. For this reason, we want to investigate in this study how useful the UA is on today’s web, more than 25 years after it was introduced, because if it is possible to remove it, it would seriously affect the capacity of browser fingerprinting to track users online.

Restricting the User Agent. In the past, developers of the Safari browser froze the UA to reduce web compatibility risks and to prevent the use of UA for browser fingerprinting [11]. The community quickly reported page breakages, due to lack of UA information. Building on past experience, the W3C Community Group introduced UA client hints to reduce UA granularity [53]. Developers of Chrome browser built on that work to implement UA reduction and has since been progressively released in new Chrome versions [53]. Developers at Mozilla/Tor have been working on browser fingerprint resistance and have collected several reports of web page breakages [47]. However, no study has been conducted to study the impact of frozen or reduced UA on the web.

2.2 Measuring Web Page Similarities

Web similarity was previously studied for content categorization, anti-phishing, browser performance optimization, user experience, web archiving, and crawling strategy [1, 8, 15, 29, 44, 51, 52]. Those studies were limited to individual comparisons of text, visual rendering, or HTML structure. Tombros *et al.* studied factors that determine web page similarity by evaluating the effectiveness of HTML structure and content [46]. Hashmi *et al.* worked on QLUE, a visual comparison tool that evaluates the content and functionality of web pages using structural similarity [20]. QLUE takes an excessive amount of time to produce results, hindering its scalability, so a novel approach was required to compare visual rendering in the wild. No work has been done before to develop compound metrics

to fully understand the similarity of web pages. Our paper is the first that provides similarity metrics to explore the impact of UA on the web.

3 UA-RADAR: MEASURING WEB SIMILARITY IN THE WILD

3.1 Overview

Given the evolution of web technologies, measuring the similarity of two web pages is a complex task that requires considering multiple dimensions. While a raw web page is an HTML document sent by the web server, web pages include *Cascading Style Sheets* (CSS) that describe how the page must be rendered in the browser, and *JavaScript* (JS) programs that the browser relies on to interact with the user and inject dynamic behavior into the web page. Therefore, an effective comparison of two web pages requires exploring multiple dimensions of similarities to better detect the occurrences of any difference. To that end, we introduce a similarity radar that relies on the following dimensions: 1) the *HTML markup* which represents the structure of the page with only the nodes of the DOM tree, 2) the *HTML content* which contains all the content of the nodes of the DOM tree, 3) the *JavaScript* code present in the page, 4) the CSS code included in the page, and 5) the *visual rendering* or visual similarity between two pages. Separating a page along these dimensions ensures the comparison can be articulated around meaningful and logical parts of a page. This helps us pinpoint more easily the source of a difference and it also facilitates the comparison, as each dimension can have its own comparison algorithm since JS code behaves differently from CSS code and regular textual content. Finally, visual rendering was added to understand if providing a UA header with only the string "None", which contains no specific device information would break a website or not.

In Figure 1, we present the similarity radar with three colored pentagons, each representing a comparison between a standard browser and its corresponding None browser counterpart. Specifically, we compare Chromium versus Chromium-None (CCN), Firefox versus Firefox-None (FFN), and WebKit versus WebKit-None (WWN). The vertices of each pentagon are anchored to the similarity scores for the five dimensions discussed above: HTML structure, HTML content, visual rendering, JavaScript, and CSS. The further a vertex is from the center of the chart, the higher the similarity score. As such, a pentagon that covers a larger area within the chart represents a higher degree of overall similarity. In the scenario where all three pentagons overlap towards the 100% mark on all five dimensions, the radar charts reveal that the None-browsers are highly similar to their standard counterparts. This indicates that the UA has a marginal impact on the web page. On the other hand, if there are deviations from the overlap scenario such as the FFN where the pentagon is smaller than the others, it suggests that Firefox is impacted when the UA is not known.

Removing dynamic content. To understand the impact of different UA, it is important to filter out the content that may differ because a page may have been visited at a different time of the day (like different articles shown on a news website) or that has been personalized based on user preferences. To achieve this goal, we extract a backbone for each visited web page by comparing the page

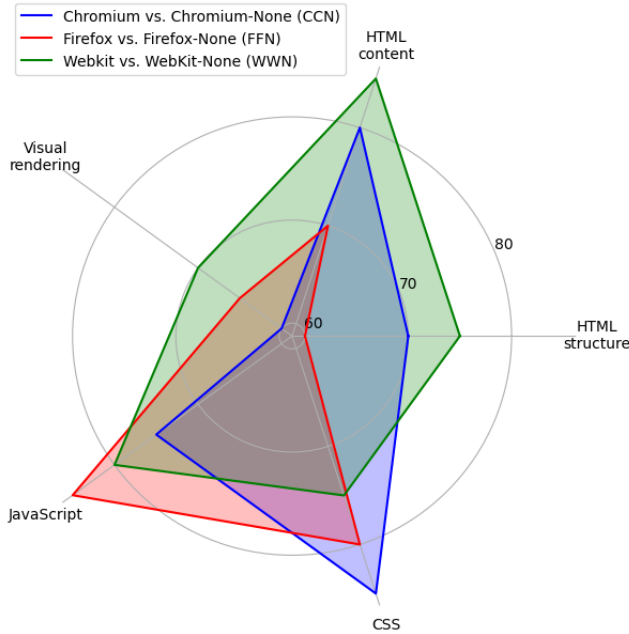


Figure 1: Similarity radar for a web page: the above represents the similarity between standard browsers and their None counterparts when accessing the home page of www.academiabarilla.it. Each colored pentagon corresponds to a single comparison, and its vertices represent the similarity scores across five dimensions: HTML structure, HTML content, visual rendering, JavaScript, and CSS. Overlapping pentagons near the 100% mark indicate a marginal impact of the UA on the web page.

with itself collected from two distinct crawls. This way, the dynamic content is revealed and can be excluded from our similarity analysis. For example, if we visit a web page twice using a standard browser (UA) and download its resources as W_1 and W_2 , A is the backbone of the visited web page such that every resource in A belongs to both W_1 and W_2 . Similarly, when investigating the impact of UA, the same page is visited twice with a None-browser (UA'), and resources are downloaded as W_3 and W_4 . B is the backbone for the web page visited using UA' , such that every resource in B belongs to both W_3 and W_4 . Finally, a comparison between A and B results in a similarity radar for the web page visited using UA and UA' (cf. Fig. 2).

3.2 Implementation Details

Our methodology for measuring web page similarity is a combination of both existing tools and research to evaluate the similarity of the HTML structure, HTML content, JavaScript, and CSS. To measure the visual similarity of web pages, we propose a novel approach for comparing web page screenshots using traditional image processing techniques. This comprehensive approach provides an in-depth understanding of web page similarity in the wild.

Removing dynamic content. Every browser uses *document object model* (DOM) to hierarchically organize nodes of the HTML document as a tree that renders in the browser. This allows structural comparison of 2 DOM trees to capture the similarity of the HTML structure of 2 web pages. Previous works have studied document structural similarity algorithms based on tree edit distance, tree matching, and various approximation techniques [4, 6, 40]. We use SFTM, a DOM tree matching tool, to measure the similarity of the HTML structure of two web pages in the wild [5]. We choose SFTM because, in contrast to other tree-matching algorithms, the algorithm behind it efficiently processes any valid DOM tree in microseconds, not dozens of seconds. To assess the similarity of the HTML structure of 2 web pages, we extract the DOM trees from the HTML documents and feed them to SFTM. SFTM then builds a graph between the two DOM trees with edges representing edit operations on the nodes. We then create labels for the insertion, deletion, and replacement operations on the nodes in the graph. The output of the process is an object containing the number of edges ($|E|$) and the number of edit operations ($|C|$).

A similarity score (S_1) is then computed as $S_1 = |C|/|E|$.

Comparing the HTML content. We use the Diff Match Patch library [18] to compare the similarity of HTML content over other text comparison tools for several reasons. Firstly, it has a high level of accuracy in detecting similarities between text snippets even in the presence of minor variations, such as white space or case sensitivity. Additionally, the library has the capability to handle long text sequences efficiently, making it ideal for comparing HTML content which can often be lengthy. Finally, the library offers a flexible API, allowing for customization of the comparison process to meet specific requirements, such as ignoring HTML markups in this context. These features make Diff Match Patch an ideal choice for evaluating the similarity of HTML content. The Diff Match Patch library implements Myer’s diff algorithm [33]. To assess the similarity of the HTML content of 2 web pages, we feed the raw content of the HTML documents to Diff Match Patch. The library returns a graph of edges (E) with edit operations on the nodes. Diff Match Patch uses the Levenshtein distance (d) to compute the number of edit operations between 2 streams of characters from the HTML documents. A similarity score (S_2) is then computed such as $S_2 = d/|E|$ [54].

Comparing JS & CSS. To assess the similarity of JS or CSS on 2 web pages, we need to first build an *abstract syntax tree* (AST) from each JS script and CSS stylesheet. We apply a *Locality-Sensitive Hash* (LSH) function on the content of each file to determine specific files with changes [10]. Once we extract ASTs to compare, similarly to the process of comparing the HTML structure, we conduct a tree-matching process to obtain a graph of edges and edit operations on nodes of the AST trees. We use GUMTREE, an AST diff tool that allows a plug-and-play of language parsers, to compare the AST trees [16]. We then create labels for the insertion, deletion, and replacement operations on the nodes in the resulting graph. The output of the process is an object containing the number of edges ($|E|$) and the number of edit operations ($|D|$) reported by GUMTREE. A similarity score (S_3) is then computed such as $S_3 = |D|/|E|$.

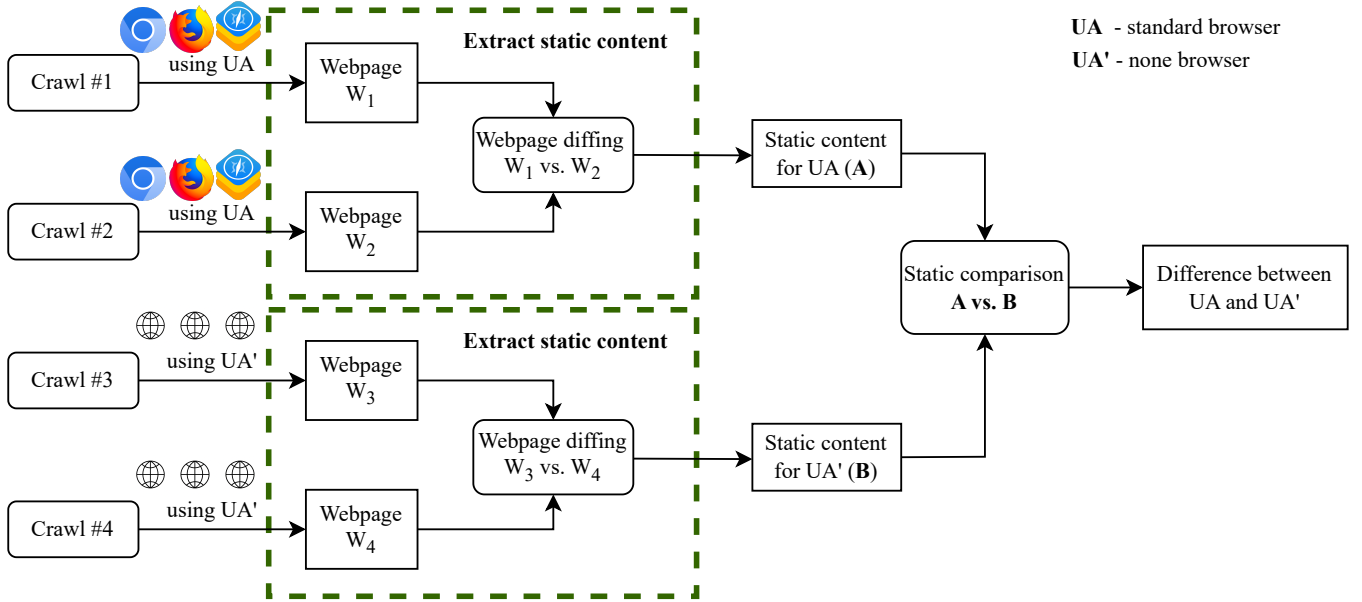


Figure 2: Highlighting web page similarity: standard browser (UA) versus None-browser (UA'). We crawl each web page twice using standard browsers (Chromium, Firefox, WebKit) and their None-browser counterparts. The dual crawl allows us to filter out dynamic content and focus on the static content of the web page, thereby eliminating potential bias in our analysis.

Subsequently, we execute a static comparison between standard and None-browsers' pages to identify UA-attributable differences, thereby facilitating the computation of similarity scores.

Comparing visual rendering. State-of-the-art image comparison algorithms are based on perceptual hashing, histogram, or by looking at pixel-by-pixel changes [12, 25, 43, 48, 49]. While those algorithms can detect the smallest difference when comparing pictures or screenshots of web pages, they fail to capture more macro changes, like text changes, broken links, or missing images. We introduce a novel approach to compare the visual rendering of web pages based on the Canny edge detection algorithm to detect any object or shape in the screenshot, which can represent text, multimedia content, and visual sections of the web page regardless of its size [7]. Our algorithm computes the number of edges (contours) in a screenshot of the web pages, hence calling it contour-based analysis.

We rely on OpenCV [36] to retrieve contours from an image. OpenCV implements the shape analysis algorithm by Satoshi *et al.* [45]. For better accuracy, we convert the original screenshot into a binary image and then find the contours in the image. Using the contours in the image, we compute the areas of the contours and their *moments*. In OpenCV, moments are the average of the intensities of an image's pixels. The area of a contour gives it relevance compared to other contours in the image while the moment helps us determine the difference in the same contour. For example, web pages on news websites often have the same contours, but with different text and photos in the same placeholder. Using moments helps us determine if the content within the same contour has changed. Algorithm 1 shows the steps we take to compute the properties of our image contour analysis, where s is the file path of the screenshot, while $cv::cvtColor$, $cv::findContours$, $cv::contourArea$,

$cv::boundingRect$ are arrays computed using OpenCV functions. The image contour properties that we compute are the number of contours ($|C|$), the weighted aggregate of contour areas (A), and the weighted aggregate contour moments (M).

Algorithm 1 Contour properties: this algorithm takes an input image s , converts it to a grayscale image g , finds contours C from the grayscale image, stores the area of each contour in Y , calculates the bounding rectangle Z , and computes the weighted areas A and moments M of the contours.

```

1: function FINDCONTOURPROPERTIES( $s$ )
2:    $g = cv::cvtColor(s)$   $\triangleright$  Convert image  $s$  to grayscale image  $g$ 
3:    $C = cv::findContours(g)$   $\triangleright$  Find contours  $C$  from image  $g$ 
4:    $Y = cv::contourArea(C)$   $\triangleright$  Store area of each contour in  $Y$ 
5:    $Z = cv::boundingRect(C)$   $\triangleright$  Bounding rectangle  $Z$ 
6:   Let  $A = 0, M = 0$   $\triangleright$  Weighted areas  $A$  & moments  $M$ 
7:   for  $i = 1$  to  $|C|$  do
8:      $A = A + Y_i^2 \div Y$ 
9:      $M = M + Z_i^2 \div Z$ 
10:  end for
11:  return  $<|C|, A, M>$ 
12: end function
  
```

Since the contour properties (C , A , and M) are heterogeneous, when comparing two screenshots to find their similarity, we compute the geometric mean (GM) of the contour properties of each screenshot. Finally, we compute the visual similarity score (S) as



Figure 3: Contour-based visual analysis: the figure illustrates the process of contour-based analysis on a screenshot taken from www.academiabarilla.it. The top image represents the original screenshot, while the bottom image shows the identified contours (edges), representing different objects and shapes within the web page. This technique enables a comparison of visual rendering, capturing significant changes such as text modifications, broken links, or missing images.

the ratio of the absolute difference of the geometric means (GM_1 and GM_2) to their arithmetic mean such that:

$$GM = \sqrt[3]{|C| \times A \times M} \quad (1)$$

$$S = \frac{|GM_1 - GM_2|}{(GM_1 + GM_2)/2} \quad (2)$$

4 EXPLORING THE IMPACT OF UA CHANGES

To explore the impact of UA on the web, we crawled websites with the default HTTP's UA header and with the "None" string in its place. We conducted regression tests based on the similarity radars provided by UA-RADAR. We then analyzed the edit operations for the dimensions in each test to determine if the observed changes were due to the removal of identifying information in the UA or not. We also explore in this section why those changes occurred.

4.1 Crawl Description & Statistics

We used a web testing and automation framework called *Playwright* to instrument standard browsers, namely Chromium (C), Firefox (F), and Safari (with the WebKit engine W) [32]. To instrument the None browsers, we modified the HTTP request-header field User-Agent of the standard browsers and changed it to the word "None". We also modified information that identifies the browser in the Navigator objects of the standard browsers. In particular, we changed `navigator.appVersion`, `navigator.platform`, `navigator.userAgent`, and `navigator.vendor` and placed the word "None" on each of those properties. Furthermore, to avoid our modified browsers from being detected as bots, we set `navigator.webdriver` to false. Detailed lists of navigator properties exposed on all browsers during the crawl are available in the artifacts mentioned in Section A. We called the resulting browsers after their modified versions: *Chromium-None* (CN), *Firefox-None* (FN), and *WebKit-None* (WN). In the end, we ran the crawl with 6 browsers in total.

After preparing the browsers to be used, we decided on how to run the crawl. We used the TRanco list to choose the domains to crawl [39]. We chose the TRanco list as its ranking of website popularity surpasses other sources of web traffic analysis [38]. Nevertheless, previous studies have expressed concerns about the methodology used to create popular lists, such as TRanco and their representativeness [42]. For that reason, we randomized our crawl of domains on the TRanco list until we reached the limit of our computing resources. We finally crawled homepages of 12,000 domains with 1,765 in the Top 10k domains, 6,036 between the Top 10k and Top 100k, and 4,199 between the Top 100k and Top 1M. Aqeel *et al.* have also questioned the representativeness of measurement studies that rely only on landing pages and no internal pages, citing a difference in structure and content between the landing page and the internal pages[3]. This was not a concern for our study as our objective was to analyze the impact of restricting the UA without being specific on the type of structure or content.

We crawled the homepage of each domain four times with each browser: twice (for self-comparison to remove dynamic content) before the execution of JS to study differential serving, and twice after the execution of JS to study content adaptation. This represents

24 visits per homepage in total. For differential serving, we waited for the *domcontentloaded* event to be fired before saving on disk the complete HTML document with all first and third-party JS and CSS files. For content adaptation, we waited 15 seconds after the *load* event was fired to save everything on disk. To avoid being rejected due to a high number of requests, our crawler sent exactly one request to one domain with one browser at a time, and only multiple requests to multiple domains in parallel. The crawl ran for 1 month. A repository for the dataset of this paper is available in the artifacts mentioned in Section A.

If all 24 requests were not successfully crawled with the data correctly saved, the crawled domain was ignored and all downloaded resources for the crawled web page were deleted on the disk. In the end, we successfully crawled and saved data for 270, 048 web pages from 11, 252 domains. We stored 5.85 Terabytes of compressed files on the disk. Table 1 summarizes the resources we saved on the disk during our crawl. It is worth noting that JS takes most of the resources on the Internet today with 73% of the downloaded files and 80% of disk space in our dataset.

Table 1: Summary of crawled resources

Resource type	Number of files	Resource size
HTML	180, 032	17 GB
JavaScript	73, 573, 872	4, 705 GB
CSS	27, 060, 120	959 GB
Screenshots	180, 032	167 GB
Total:	100, 994, 056	5, 848 GB

4.2 Empirical Results & Findings

In this section, we use the following notations: CCN for the comparison between pages from Chromium against Chromium-None, FFN for Firefox against Firefox-None and WWN for WebKit against WebKit-None.

Differential serving. The average similarity scores before JS is executed are 100% for all the tested browsers (CCN, FFN, and WWN) on HTML structure, HTML content, JS, and CSS (cf. Figure 4a). One takeaway is that web servers reply to all HTTP requests with the same HTML document, regardless of the fact that the UA in the HTTP request header is known or not. This is possible because browsers adopt the same standards, such as responsive web design to adjust the rendering of web pages to browsing environments. That means that, nowadays, websites focus on consistent user experience across devices and browsers rather than device-specific content. The fact that UA is no longer the sole factor in determining the content served makes them less relevant and hence removing the significance of the UA can reduce the attack surface and improve the privacy and security of users.

Content adaptation. When JavaScript is executed, the average similarity scores remain 100% on JS and CSS. However, there are changes in visual rendering, the HTML structure, and the HTML content. 158 out of the 11, 252 domains are not 100% visually similar for at least one of CCN, FFN, or WWN. Going with the same logic,

955 are not 100% similar on both the HTML structure and HTML content for at least one of CCN, FFN, WWN. We looked up the 158 domains with at least one difference and found that all those domains also have at least one difference on both HTML structure and HTML content for at least one of CCN, FFN, or WWN. We then established that, out of 11, 252 domains, 955 are changed by the lack of a known UA or other identifying information. That is, 8.4% of our dataset was dependent on the UA. Table 2 lists the Internet categories to which the UA-dependent domains belong. We used McAfee SmartFilter to obtain the Internet categories of the UA-dependent domains [30].

Table 2: Internet categories of the UA-dependent domains

Category	Number of Domains
News	180
Internet Services	156
Business	128
Online Shopping	109
Marketing	106
Blogs	89
Education	71
Entertainment	68
Information	38
Finance	10
Total	955

Table 3: Summary of problem severity levels for UA-dependent domains

Category	Number of domains
IRRITANT	225
MODERATE	131
NO PATTERN	6
SEVERE	526
UNUSABLE	67
TOTAL	955

The last 5 radars depicted in Figure 4 provide a category-wise breakdown of the average similarity scores between standard browsers and their None-browser counterparts for the top 5 website categories in our data set, namely: news, internet services, business, online shopping, and marketing. In the category "news" (cf. Figure 4b), the HTML content and visual rendering dimensions have lower similarity scores compared to other dimensions. This suggests that, in this category, the absence of a known UA tends to influence the visual presentation and HTML content of the web pages more significantly. For the "internet services", "business", and "marketing" categories (cf. Figures 4c, 4d, and 4e), all three comparisons (CCN, FFN, and WWN) show similarity scores gravitating towards 100% across all dimensions. This indicates the similarity of HTML content and visual rendering, irrespective of the browser's UA. Therefore, we can infer that the impact of the UA on these categories is marginal. The category "online shopping" (cf. Figure 4f),

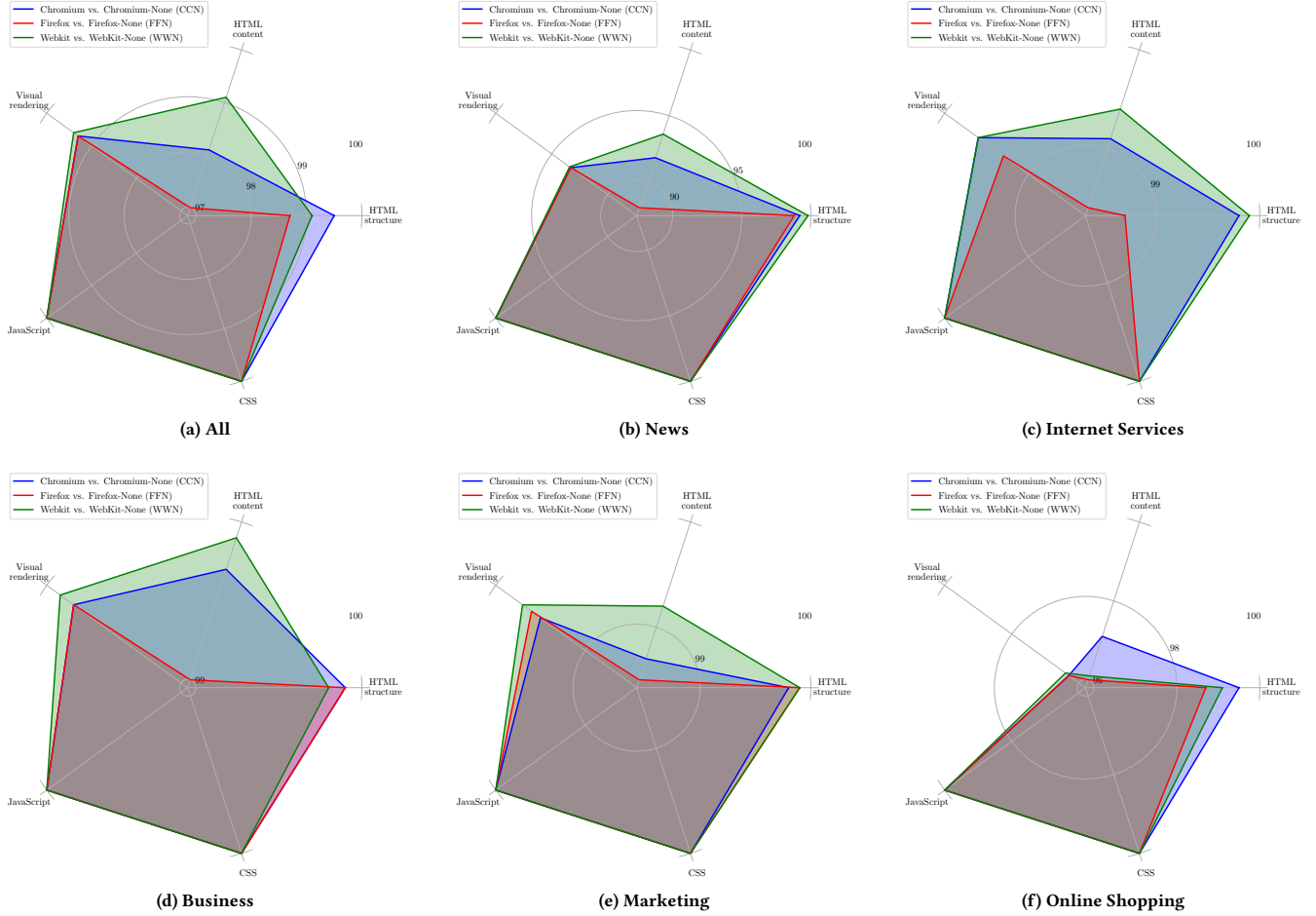


Figure 4: Average similarity scores across website categories: this figure illustrates the average similarity scores between standard browsers and their None-browser counterparts for the top five categories in our dataset.

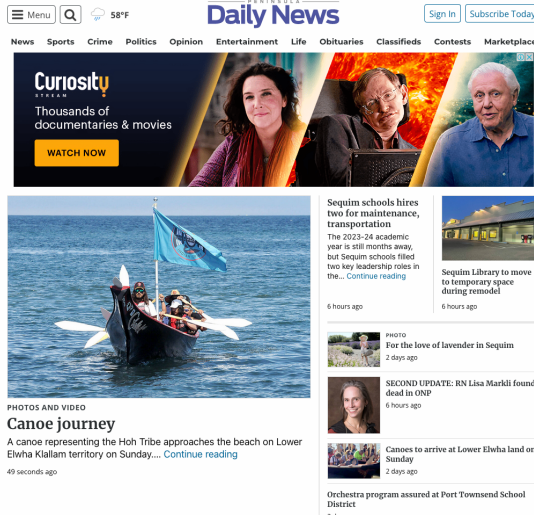
however, presents a contrast. The HTML content and visual rendering dimensions have lower similarity scores, close to the category "news", but we observe a departure from the pattern observed in other categories. While, the similarity scores for the WebKit browser usually top the charts, followed by Chromium, in the category "online shopping" the HTML content similarity score for WebKit drops dramatically, coming closer to Firefox. Additionally, the visual rendering is impacted across all browsers. This may suggest that online shopping websites employ more complex or diverse techniques for content adaptation based on UA, which could potentially be linked to the need for enhanced user experience or functionality specific to the website's purpose.

The consistent performance of WebKit, then Chromium across categories prompts further exploration. One plausible explanation could lie in the rendering engine used by the browsers. Both browsers use Blink, however, WebKit consistently outperforms Chromium in our analysis. While this difference could stem from how each browser integrates and uses the Blink engine, the underlying reasons for this consistent trend are not immediately apparent

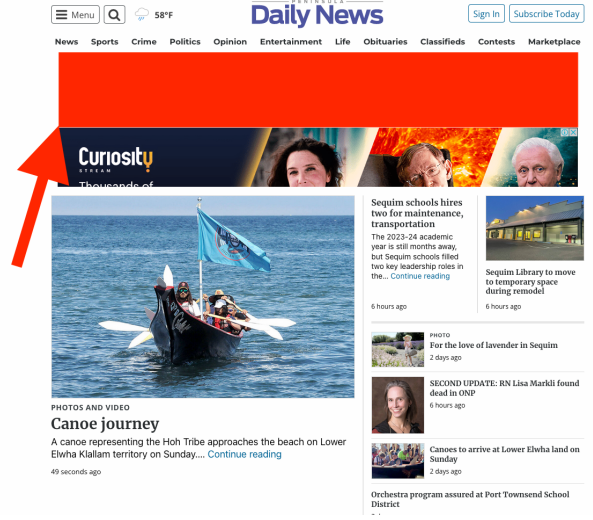
from our study and would require further investigation. Such research could provide insights into how browser architecture and rendering engines influence content adaptation.

Changes created by different UA and their causes. Due to the intensive manual efforts required to analyze the changes, out of the 955 domains that changed because of the None-browser, we manually analyzed the changes for 204 domains. We detected 10 patterns of the impact of those changes and used the 10 patterns to apply heuristics to the rest of the data set in order to classify the severity of the impact on the usability of the web pages.

To build that classification, we borrowed the taxonomy of problem severity scale in usability by Rubin *et al.* [41]. Algorithm 2 details the steps in conducting the heuristics for that classification. The algorithm takes six inputs: C , CN , F , FN , W , and WN , which represent the used browsers: Chromium, Chromium-None, Firefox, Firefox-None, WebKit, and WebKit-None, respectively. It then performs static comparison operations between each standard browser and its None counterpart to determine the differences, denoted as ΔCCN , ΔFFN , and ΔWWN .



(a) Standard browser - normal margin rendering



(b) None browser - failure of margin collapse

Figure 5: Comparison of web page rendering with standard and none browsers, illustrating a 'severe' problem severity case where a failure of margin collapse occurs in the none browser. The affected area is highlighted in red.

If all three differences are identical, the algorithm then computes the differences between every pair of standard browsers, denoted as ΔCF , ΔCW , and ΔFW . It also assigns the three differences from the standard to None comparisons to the list R , and the differences from the standard to standard comparisons to the list N . Following this, the algorithm checks for specific CSS and HTML properties and attributes in the differences. These include the CSS properties "margin-top, bottom", "white-space: wrap", "page-break-before, after", any CSS attributes associated with HTML tags, the image source attribute, and the width or height attributes of iframes. For each of these, if they are found in the differences, the algorithm returns a corresponding impact statement. To select the properties and attributes that the algorithm used, we conducted a manual analysis of 100 websites to build a list of HTML and CSS properties that cause changes in the web page when the UA is not known. Afterward, we utilized that list to classify the levels of problem severity identified by our change impact analysis.

```
1 function rn(a, b, c, d) {
2   0(a.K, {
3     transition: c / 1E3 + "s",
4     "transition-timing-function": d,
5     "margin-top": b
6   })
7 }
```

Listing 1: Example of unintentional restriction: a function that relies on known UA (parameter b) in a script from Google Ad Manager.

Additionally, if a disabled or inactive tag is found, or any instance of CAPTCHA, HTTP 403 error is detected, the algorithm will return the respective impact statements. If the differences in R are not identical to this N , the algorithm returns "content restriction". If none of the specific properties or attributes is found, it returns "no pattern". If the three differences are not identical, the algorithm

checks if each difference is unique and returns "no impact" if that is the case. Otherwise, it returns "no impact".

Below are the definitions of the problem severity categories that we use:

- (1) Irritant: The problem occurs only intermittently, can be circumvented easily, or is dependent on a standard that is outside the product's boundaries. Could also be a cosmetic problem.
- (2) Moderate: The user will be able to use the product in most cases, but will have to undertake some moderate effort in getting around the problem.
- (3) Severe: The user will probably use or attempt to use the product here, but will be severely limited in his or her ability to do so.
- (4) Unusable: The user is not able to or will not want to use a particular part of the product because of the way that the product has been designed and implemented.

It should be noted that we do not address the behaviour and interaction with the functionality of the web page, so the use of the word "UNUSABLE" in the problem severity scale by Rubin *et al.* should not create confusion. Table 4 shows the classes of the impact of the changes and the severity of the impact. On 425 out of the 955 domains (44.5%), the impact of browsing those domains with a None-browser was spacing issues (failure of margin collapsing, failure of soft-wrap, and unnecessary blank lines), while on 515 domains, the impact was driven by CSS issues. Table 3 shows the distribution of problem severity for the 955 domains, revealing that just 7% of the domains were unusable when we browsed them using a None-browser.

Some HTML elements change while the browser adjusts the web page to an unknown UA. These changes only occur after the execution of JavaScript. Therefore, the cause of the changes is

Table 4: Change impact analysis of the 955 UA-dependent websites: this table details the specific changes detected, their associated impact, the problem severity level, and the number of occurrences, providing a comprehensive overview of how changes in the UA affect different aspects of the web page.

Change	Impact	Severity	Occurrences
CSS property: margin-{top,bottom}	Failure of margin collapsing	SEVERE	252
{ $\Delta CCN \neq \Delta FFN \neq \Delta WWN$ } & { $\Delta CF \neq \Delta CW \neq \Delta FW$ }	No impact	IRRITANT	225
Missing image SRC reference	Failure of lazy loading	SEVERE	101
CSS property: white-space: wrap	Failure of soft-wrap	SEVERE	99
Change of CSS attribute(s)	Change of inline CSS	MODERATE	83
CSS property: page-break-{before,after}	Unnecessary blank lines	SEVERE	74
iFrame width height	Displaced iframe	MODERATE	48
{ $\Delta CCN = \Delta FFN = \Delta WWN$ } \neq { $\Delta CF = \Delta CW = \Delta FW$ }	Content restriction	UNUSABLE	38
CAPTCHA or 403 Error or Browser Error	Browser not identified	UNUSABLE	22
CSS :disabled :inactive	Disabled component	UNUSABLE	7
No pattern	-	-	6

located in JS scripts on the web pages. Hence, we can say that JS is the cause of the changes and that CSS and HTML are affected by those changes. Looking at the content of JS scripts on the web pages, we did not find a pattern of similar instructions in the scripts. However, the sources of the scripts produced a pattern. 76% of the SEVERE issues were caused by third-party scripts from Google Ad Manager, while 93% of all changes in the HTML content comparison were caused by third-party scripts from ad domains. The remaining 7% of domains responsible for UA changes were from bot detection and *content delivery networks* (CDN) websites.

Current ad-displaying scripts rely on known browsers, so the use of None-browsers creates invalid references. For example, in Figure 5, we showcase a prominent instance of a 'severe' problem severity case that we investigated during our study. In the first subfigure, we see the web page as accessed through a standard browser, where the rendering and layout are as intended. However, the same web page, when accessed through a None browser, as shown in the second subfigure, experiences a failure of margin collapse, a fundamental aspect of CSS layout. This failure results in a distortion in the web page's layout, as highlighted in red in the figure. Upon investigating the cause of this issue, we found a script from Google Ad Manager that was affecting the rendering of the page in None browsers. As presented in Listing 1, the function 'rn(a, b, c, d)' applies a CSS transition and a top margin to an element based on the known UA (parameter 'b'). When the User Agent is not recognized, as in the case of a None browser, the function fails to apply the intended styles, causing the observed layout distortion.

```

1 function H1() {
2   var a, b;
3   return "function" === typeof(null == (a = E.navigator)
4     ? void 0 : null == (b = a.userAgentData) ? void 0
5     : b.getHighEntropyValues)
6 }
7 ...
8 H1() ? (d(), t(r.linkAttribution)) : r.enableRecaptcha &&
9   p("require", "recaptcha", "recaptcha.js");

```

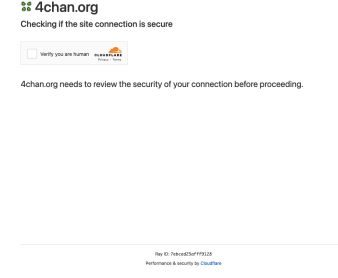
Listing 2: Example of intentional restriction: a function that runs a CAPTCHA test when the UA is not recognized.

We classified such cases that cause usability issues due to code written to acknowledge the UA as an unintentional restriction. However, the cause of the remaining 7% of the issues ranked as "UNUSABLE" was intentional. For example, In Figure 6, we illustrate a case of the "unusable" problem severity level. The first subfigure portrays a standard browser smoothly accessing a web page normally without any issues. The second subfigure illustrates a None browser attempting to access the same web page but being intentionally restricted. This is an example of the "unusable" severity level, where the user's ability to access the web page is hindered due to the intentional restriction applied when an unrecognized UA is detected. This restrictive behavior is commonly driven by JavaScript scripts embedded in the website that use the UA string to dictate access or modify the user's experience. As shown in Listing 2, some scripts initiate a CAPTCHA test or a similar challenge when they fail to recognize the UA. In the case of a None browser, whose UA string is not recognized, this results in an intentional restriction, preventing the user from accessing the site's content.

In Figure 7, we present the distribution of problem severity across the top five website categories: news, Internet services, business, online shopping, and marketing. The heat map allows us to observe the prevalence of the different severity levels, namely, irritant, moderate, severe, and unusable, across these categories. The color intensity in each cell of the heat map is proportional to the number of websites in a category that falls under a particular problem severity level. Lighter colors indicate a lower count, while darker colors represent a higher count. The heat map provides a visual summary of our findings, revealing the extent to which different website categories are impacted by changes in the UA. For example, we can observe that the 'Severe' problem severity level is particularly prevalent in the 'news' and 'online Shopping' categories. Conversely, the 'Internet services' and 'marketing' categories have a substantial number of websites with 'irritant' or 'moderate' problem severity levels. The final observation is the pattern of the 'unusable' severity level, where the majority of occurrences are concentrated in the 'Internet services' and 'online shopping categories'.



(a) Standard browser - normal page access



(b) None browser - access intentionally restricted

Figure 6: Example of "unusable" problem severity: access to the web page is intentionally restricted when using a None Browser.

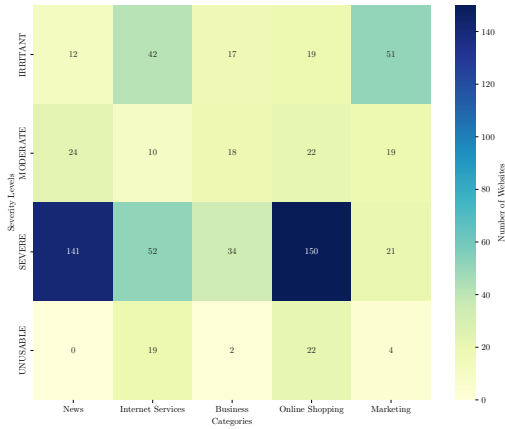


Figure 7: Problem severity distribution across website categories: this heat map depicts how changes in the UA impact different website categories, highlighting the prevalence of problem severity levels in each category.

Impact of removing identifying information from the UA.

Firstly, the UA request-header field in the HTTP request has no impact on the web server's response. Secondly, the navigator.userAgent is used marginally for ads, bot detection, and CDN services, and the use of UA, in this case, causes usability problems of different severity. Additionally, we found that browsers could still determine that a None-browser is related to its descendant standard browser. The usability problems experienced due to the removal of identifying information in the UA could be fixed by adopting a feature detection approach in determining the browser in the case of bot detection or by adopting a browser-agnostic approach in writing the code in the case of ads and CDN services. This approach ensures that the user's privacy is protected while also promoting a secure web browsing experience.

5 DISCUSSION

While useful when it was introduced 3 decades ago, our study shows that the User-Agent HTTP header, which contains precise device information, has stopped being relevant on today's web. With the crawls that we performed, our results highlight that web servers do not adapt their response anymore based on the provided HTTP UA header. By providing different UA, all responses we collected for a single web page were identical and the only differences we observed were done by scripts that would parse the provided user agent at runtime. Then, the data we obtained during our crawls highlight the two following key insights:

- All the standardization efforts pursued by the major web actors have had a real positive impact on the web. Browsers have become robust enough that they do not need web pages tailored for them. The browsers used on the market today implement the same set of features and provide a near-identical experience when it comes to rendering pages.
- There are no major hurdles to retiring the historical HTTP User-Agent header and transitioning towards a less-granular solution like UA Client hints [53]. As mentioned in 2.1, the HTTP User-Agent header contributes a lot to the field of browser fingerprinting as it is one of the top attributes revealing the most information. Without it, the privacy of web users would be severely improved as there would be a lot fewer leaks of precise and unique information on the web.

6 IMPACT OF NONE-BROWSERS ON WEB PRIVACY

UA strings are a critical component in various tracking techniques, including browser fingerprinting, posing a significant concern for web privacy [14]. In our study, we first examined the impact of None-browsers on web page usability. To understand the potential implications on tracking techniques, we analyzed how domains in our data set accessed UA information via the JavaScript API. Specifically, 3,772 domains out of 11,252 access the UA via the JavaScript API, a common method used in browser fingerprinting [35]. Cross-referencing our dataset with a list of known trackers from uBlock Origin (uBO) [21], we found that 612 domains (5.4%)

Algorithm 2 Change impact analysis: this algorithm assesses the impact on a web page when changes occur due to the use of a None-browser. It evaluates the differences detected in the static comparison (as illustrated in Figure 2) of both standard and None-browsers. Subsequently, these differences are categorized based on specific patterns that we identified during manual analysis.

```

1: function FINDCHANGEIMPACT( $C, CN, F, FN, W, WN$ )
2:   Let  $\Delta CCN \leftarrow \text{StaticComparison}(C, CN)$ 
3:   Let  $\Delta FFN \leftarrow \text{StaticComparison}(F, FN)$ 
4:   Let  $\Delta WWN \leftarrow \text{StaticComparison}(W, WN)$ 
5:   Let  $\Delta CF \leftarrow \text{StaticComparison}(C, F)$ 
6:   Let  $\Delta CW \leftarrow \text{StaticComparison}(C, W)$ 
7:   Let  $\Delta FW \leftarrow \text{StaticComparison}(F, W)$ 
8:   if  $\Delta CCN = \Delta FFN \ \& \ \Delta FFN = \Delta WWN$  then
9:     Let  $R \leftarrow [\Delta CCN, \Delta FFN, \Delta WWN]$ 
10:    Let  $N \leftarrow [\Delta CF, \Delta CW, \Delta FW]$ 
11:    if  $\text{margin}\{-\text{top}, \text{bottom}\} \in \Delta CCN$  then
12:      return "Margin collapsing fail"
13:    else if  $\text{white-space: wrap} \in \Delta CCN$  then
14:      return "Soft-wrap fail"
15:    else if  $\text{page-break-before, after} \in \Delta CCN$  then
16:      return "Unnecessary blank lines"
17:    else if  $\text{<tag css>} \in \Delta CCN$  then
18:      return "Inline css changes"
19:    else if  $\text{<img src>} \in \Delta CCN$  then
20:      return "Lazy loading fail"
21:    else if  $\text{<iframe width|height>} \in \Delta CCN$  then
22:      return "Displaced iframe"
23:    else if  $\text{<tag inactive|disabled>} \in \Delta CCN$  then
24:      return "Disabled component"
25:    else if  $\text{CAPTCHA|403|error} \in \Delta CCN$  then
26:      return "Browser not identified"
27:    else if  $R \neq N$  then
28:      return "Content restriction"
29:    else
30:      return "No pattern"
31:    end if
32:  else
33:    Let  $x \leftarrow \Delta CCN \neq \Delta FFN \neq \Delta WWN$ 
34:    Let  $y \leftarrow \Delta CF \neq \Delta CW \neq \Delta FW$ 
35:    if  $x \ \& \ y$  then
36:      return "No impact"
37:    else
38:      return "No pattern"
39:    end if
40:  end if
41: end function

```

out of the 11,252 were on the uBO list. 38.3% of these trackers were affected by changes in the UA, suggesting that many trackers can operate without issues in the face of None-browsers or may be using other methods beyond UA strings to track users. Our findings also highlight the current state of web practices. Only 129 domains out of 11,252 contained Accept-CH response headers, suggesting that the use of Client Hints for content adaptation is not yet widespread [31]. Moreover, 584 domains return a Vary header that indicates

Domain category	Number of Domains
Total domains analyzed	11,252
Domains accessing UA via JS API	3,772
Domains with Vary UA-related header	584
Domains listed on uBO	612
UA-dependent domains	955

Table 5: Web privacy implications of UA usage: this table presents an analysis of domains based on their interaction with UA.

their use of UA. Finally, our study suggests that the majority of the web remains accessible even without UA information, with only 7% of the domains becoming unusable when browsed using a None-browser. This could encourage further adoption of None-browsers, thereby increasing user privacy.

7 THREATS TO VALIDITY

A lot of process can run on the server-side of a website and this paper focuses on the impact of UA on the client-side. This may threaten our conclusion on the impact of the UA on the web since the server-side is also part of the web ecosystem.

Our conclusion on the impact of the UA on the web is also based on the fact that the None-browsers provided the string "None" for UA and other identifying information. Empty, *null* or *undefined* UA and other identifying information may incur more breakages and other findings.

8 CONCLUSION

In this study, we investigated the role of the User Agent in today's web by crawling We crawled 270,048 web pages from 11,252 domains with different configurations. Our data shows that websites no longer negotiate content based on the UA field in the HTTP request headers. Through JS scripts, `Navigator.userAgent` can be used for content adaptation, as few websites experience usability issues when they face an unusual user agent. However, the majority of those issues are unintentionally caused by third party scripts from ads, bot detection, and CDN services and can be fixed by writing browser-agnostic code. By cross-referencing our dataset with a list of known trackers from uBlock Origin, we discovered that a substantial number of known trackers did not change due to None-browsers, suggesting their robustness or the use of other tracking methods beyond UA strings. The main takeaway of our results is that after three decades of usage, it may be finally time to retire the HTTP User Agent header and transition towards a more privacy-preserving way of sharing device information. The UA has been abused too many times over the years to reveal information about users and sometimes even identify them by contributing to their browser fingerprinting. Removing it from today's ecosystem would be a great step forward for online privacy and would contribute greatly to reducing the clutter from legacy technology.

REFERENCES

- [1] H. Ali and H. E. Williams. What's Changed? Measuring Document Change in Web Crawling for Search Engines. In M. A. Nascimento, E. S. de Moura, and A. L. Oliveira, editors, *String Processing and Information Retrieval*, Lecture Notes

- in *Computer Science*, pages 28–42, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-39984-1.
- [2] A. Andersen. History of the browser user-agent string, 2008.
 - [3] W. Aqeel, B. Chandrasekaran, A. Feldmann, and B. M. Maggs. On Landing and Internal Web Pages: The Strange Case of Jekyll and Hyde in Web Performance Measurement. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, pages 680–695, New York, NY, USA, Oct. 2020. ACM. ISBN 978-1-4503-8138-3.
 - [4] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, June 2005. ISSN 03043975.
 - [5] S. Brisset, R. Rouvoy, L. Seinturier, and R. Pawlak. Sftm: Fast matching of web pages using similarity-based flexible tree matching. *Information Systems*, 112: 102126, 2023.
 - [6] D. Buttler. A short survey of document structure similarity algorithms. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2004.
 - [7] J. Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, pages 679–698, 1986.
 - [8] Y. Chen, W.-Y. Ma, and H.-J. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 225–233, New York, NY, USA, 2003. ACM. ISBN 978-1-58113-680-7.
 - [9] S. R. Choudhary. Detecting cross-browser issues in web applications. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1146–1148, 2011.
 - [10] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, Brooklyn New York USA, June 2004. ACM. ISBN 978-1-58113-885-6.
 - [11] J. Davis. Release Notes for Safari Technology Preview 46, Dec. 2017. URL <https://webkit.org/blog/8042/release-notes-for-safari-technology-preview-46/>.
 - [12] A. Drmic, M. Silic, G. Delac, K. Vladimir, and A. S. Kurdija. Evaluating robustness of perceptual image hashing algorithms. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 995–1000. IEEE, 2017.
 - [13] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings 10*, pages 1–18. Springer, 2010.
 - [14] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1388–1401, 2016.
 - [15] S. J. Eravuchira, V. Bajpai, J. Schönwälder, and S. Crawford. Measuring web similarity from dual-stacked hosts. In *2016 12th International Conference on Network and Service Management (CNSM)*, Oct. 2016.
 - [16] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 313–324, 2014.
 - [17] A. Gómez-Boix, P. Laperdrix, and B. Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 309–318. ACM, 2018.
 - [18] Google. `google/diff-match-patch`, Jan. 2023. URL <https://github.com/google/diff-match-patch>.
 - [19] A. Grosskurth and M. W. Godfrey. Architecture and evolution of the modern web browser. *Preprint submitted to Elsevier Science*, 12(26):235–246, 2006.
 - [20] W. Hashmi, M. Chaqfeh, L. Subramanian, and Y. Zaki. Qlue: A computer vision tool for uniform qualitative evaluation of web pages. In *Proceedings of the ACM Web Conference 2022*, pages 2400–2410, 2022.
 - [21] R. Hill. `uBlock Origin (uBO)`, Aug. 2023. URL <https://github.com/gorhill/uBlock>.
 - [22] M. Z. Islam, M. M. Hossain, S. Haque, J. Lahiry, S. A. Bonny, and M. N. Uddin. User-agent based access control for dlina devices. In *2014 6th International Conference on Knowledge and Smart Technology (KST)*, pages 7–11. IEEE, 2014.
 - [23] N. Kheir. Behavioral classification and detection of malware through HTTP user agent anomalies. *Journal of Information Security and Applications*, 18(1):2–13, July 2013. ISSN 2214-2126.
 - [24] J. Kline, P. Barford, A. Cahn, and J. Sommers. On the structure and characteristics of user agent string. In *Proceedings of the 2017 Internet Measurement Conference*, pages 184–190, 2017.
 - [25] Y. Kotsarenko. Measuring perceived color difference using yiq ntsc transmission color space in mobile applications. *Programacion Matematica y Software*, 2018.
 - [26] V. H. La, R. Fuentes, and A. R. Cavalli. Network monitoring using mmt: An application based on the user-agent field in http headers. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 147–154. IEEE, 2016.
 - [27] P. Laperdrix, W. Rudametkin, and B. Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 878–894. IEEE Computer Society, 2016.
 - [28] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14(2):1–33, 2020.
 - [29] M. T. Law, C. S. Gutierrez, N. Thome, S. Gançarski, and M. Cord. Structural and visual similarity learning for web page archiving. In *2012 10th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6. IEEE, 2012.
 - [30] McAfee. Check Single URL, 2023. URL <https://sitelookup.mcafee.com/en/feedback/url?action=checksingle&sid=BF6DB84A3A60F9AEB8F69A93DA023455>.
 - [31] R. Merewood and Y. Weiss. Improving user privacy and developer experience with User-Agent Client Hints, June 2020. URL <https://developer.chrome.com/articles/user-agent-client-hints/>.
 - [32] Microsoft. Playwright Library | Playwright, 2023. URL <https://playwright.dev/docs/api/class-playwright>.
 - [33] E. W. Myers. AnO(ND) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, Nov. 1986. ISSN 0178-4617, 1432-0541.
 - [34] J. Nejati and A. Balasubramanian. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1305–1315, 2016.
 - [35] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*, pages 541–555. IEEE, 2013.
 - [36] OpenCV. OpenCV: Structural analysis and shape descriptors, 2023. URL https://docs.opencv.org/3.4/d3/dc0/group_imgproc_shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a.
 - [37] K. Pham, A. Santos, and J. Freire. Understanding website behavior based on user agent. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1053–1056, 2016.
 - [38] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.
 - [39] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. A research-oriented top sites ranking hardened against manipulation - Tranco, 2023. URL <https://tranco-list.eu/>.
 - [40] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, pages 502–511, New York NY USA, May 2004. ACM. ISBN 978-1-58113-844-3.
 - [41] J. Rubin, D. Chisnell, and J. Spool. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Wiley, Indianapolis, IN, 2nd edition edition, May 2008. ISBN 978-0-470-18548-3.
 - [42] Q. Scheitle, O. Hohlfeld, J. Gamba, J. Jelten, T. Zimmermann, S. D. Strowes, and N. Vallina-Rodriguez. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *Proceedings of the Internet Measurement Conference 2018*, pages 478–493, Boston MA USA, Oct. 2018. ACM. ISBN 978-1-4503-5619-0.
 - [43] J. Song, Y. Yang, Z. Huang, H. T. Shen, and R. Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 423–432, 2011.
 - [44] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Workshop on artificial intelligence for web search (AAAI 2000)*, volume 58, page 64, 2000.
 - [45] S. Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.
 - [46] A. Tombros and Z. Ali. Factors Affecting Web Page Similarity. In D. E. Losada and J. M. Fernández-Luna, editors, *Advances in Information Retrieval*, Lecture Notes in Computer Science, pages 487–501, Berlin, Heidelberg, 2005. Springer. ISBN 978-3-540-31865-1.
 - [47] T. Uplift. Fingerprinting resistance, 2023. URL https://bugzilla.mozilla.org/show_bug.cgi?id=1329996.
 - [48] C. Vallez, A. Kucharavy, and L. Dolamic. Needle in a haystack, fast: Benchmarking image perceptual similarity metrics at scale. *arXiv preprint arXiv:2206.00282*, 2022.
 - [49] V. Vysniauskas. Anti-aliased Pixel and Intensity Slope Detector. *ELECTRONICS AND ELECTRICAL ENGINEERING*, Oct. 2009.
 - [50] W3C. Http request fields, 2023. URL <https://www.w3.org/Protocols/HTTP/HTREQ-Headers.html>.
 - [51] H. Wang, M. Liu, Y. Guo, and X. Chen. Similarity-based web browser optimization. In *Proceedings of the 23rd international conference on World wide web, WWW '14*, pages 575–584, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2744-2.
 - [52] L. Wenxin, G. Huang, L. Xiaoyue, Z. Min, and X. Deng. Detection of phishing webpages based on visual similarity. In *Special interest tracks and posters of the 14th international conference on World Wide Web - WWW '05*, page 1060, Chiba, Japan, 2005. ACM Press. ISBN 978-1-59593-051-4.
 - [53] A. White. User-Agent reduction, 2023. URL <https://developer.chrome.com/docs/privacy-sandbox/user-agent/>.
 - [54] L. Yujian and L. Bo. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), June 2007. ISSN 1939-3539.

A APPENDICES

Our dataset including a full list of crawled domains and their categories, a full list of navigator properties exposed during the crawl,

and other crawl and comparison data along with information on the tool we developed for UA-RADAR metrics can be found at <https://github.com/intumwa/ua-radar>