Exploring the Impact of the User's Browsing Environment on Web Privacy

Jean Luc Intumwayase

MADIS Graduate School
Inria Centre at the University of Lille
CRIStAL — Spirals research team

Exploring the Impact of the User's Browsing Environment on Web Privacy

***Jean Luc* INTUMWAYASE**

Thesis defended on December 5th 2024 for the degree of doctor of philosophy in computer science in front of a jury composed of

Supervisor & Co-supervisor

Romain Rouvoy	Professor	University of Lille
Pierre Laperdrix	Research Scientist CR	CNRS

Reviewers

Johann Bourcier	Professor	University of Pau
Nataliia Bielova	Research Director	INRIA

Examiners

Walter Rudametkin	Professor	University of Rennes (Jury President)
Aurore Fass	Associate Professor	CISPA

École Graduée MADIS
Centre Inria de l'Université de Lille
CRIStAL — Équipe de recherche Spirals

Explorer l'Impact de l'Environnement de Navigation de l'Utilisateur sur le Respect de la Vie Privée en Ligne

***Jean Luc* INTUMWAYASE**

Thèse soutenue le 5 décembre 2024 pour obtenir le grade de docteur en informatique devant le jury composé de

Directeur & Co-directeur

Romain Rouvoy	Professeur	Université de Lille
Pierre Laperdrix	Chargé de recherche	CNRS

Rapporteurs

Johann Bourcier	Professeur	Université de Pau
Nataliia Bielova	Directrice de recherche	INRIA

Examinateurs

Walter Rudametkin	Professeur	Université de Rennes (Président du Jury)
Aurore Fass	Maitre de conférence	CISPA

Abstract

The evolution of the web has followed a dual trajectory, marked by technological advancements that enhance user experience and an increase in privacy risks. User tracking techniques exploit aspects of the user's browsing environment (UBE), and this exposes users to privacy risks. Addressing these risks while maintaining seamless web functionality is essential for the future of the web. In this thesis, we investigate the impact of the UBE on web privacy by analyzing the trade-off between usability and privacy. Our aim is to enhance web privacy by minimizing unnecessary UBE information exposure while maintaining website functionality. We provide three contributions for understanding the interplay between the user's browsing environment and web privacy:

1. We present a framework to determine the relevance of information within the UBE in relation to website functionality when accessed by websites. We categorize UBE information into geolocation, device, and browser attributes, and simulate website visits using various UBE constructs through browser instrumentation. This approach allows us to systematically restrict specific UBE attributes and observe the resulting website behavior. Our methodology involves designing a web crawler that uses these different UBE constructs and collects data from websites. To quantify the impact of modifying UBE attributes, we introduce Similarity Radar, a multidimensional tool that compares web pages and computes similarity scores. We use Similarity Radar to compare website behavior in normal versus restricted environments to identify discrepancies that indicate the relevance of specific UBE attributes. This framework enables the safe restriction of nonessential UBE information, thereby enhancing user privacy.

2. We use Similarity Radar to investigate the impact of the User Agent (UA) string and associated device information on serving its original purpose of tailoring website content to various browsers. Similar to the comparison between standard and restricted UBE described previously, we crawl $270,048$ web pages across $11,252$ domains using three standard browsers and a set of 'None' browsers—clones of the standard browsers with their UA and associated information set to 'None'. We then compare the similarity between the standard and 'None' browsers. Before JavaScript (JS) execution, we observe 100% similarity, indicating that UA headers are irrelevant to website functionality today. However, after JS execution, 8.4% of crawled web

pages change. These changes are primarily driven by third-party scripts related to advertising, bot detection, and content delivery networks. Further investigating, we classify these changes based on levels of usability severity, with most being minor CSS adjustments and a smaller percentage causing significant usability issues. This indicates the obsolescence of the UA string and associated device information amidst its significance in browser fingerprinting.

3. We investigate the enforcement of cookie consent notices across different continents, focusing on how visibility and impact vary by geographic location. We use a novel automated visual detection technique based on Similarity Radar to efficiently detect consent notices. For each of the five countries—Brazil, France, Japan, South Africa, and the United States—we analyzed 14,078 websites, repeating this process three times under different interaction scenarios: first without interacting with cookie consent notices, then by accepting them, and finally by rejecting them. The results reveal disparities in consent notice prevalence, with France showing the highest visibility at 69% and Japan the lowest at 27%. We also observe that third-party cookies increase after users interact with consent notices, particularly in the US, while France maintains the lowest number of third-party cookies due to its stringent regulations. This indicates a correlation between consent notice visibility and user tracking practices.

Résumé

L'évolution du web a suivi une trajectoire double, marquée par des avancées technologiques qui améliorent l'expérience utilisateur et une augmentation des risques pour le respect de la vie privée. Les techniques de traçage des utilisateurs exploitent des aspects de l'environnement de navigation de l'utilisateur (UBE). Aborder ces risques tout en maintenant une fonctionnalité web fluide est essentiel pour l'avenir du web. Dans cette thèse, nous étudions l'impact de l'UBE sur le respect de la vie privée en ligne en analysant le compromis entre utilisabilité et respect de la vie privée. Notre objectif est d'améliorer le respect de la vie privée en minimisant l'exposition inutile des informations UBE tout en maintenant la fonctionnalité des sites web. Nous apportons trois contributions dans ce domaine :

1. Nous présentons un cadre pour déterminer la pertinence des informations au sein de l'UBE en relation avec la fonctionnalité des sites web lorsqu'ils sont consultés. Notre approche permet de restreindre systématiquement des attributs UBE et d'observer le comportement résultant du site web. Nous introduisons Similarity Radar, un outil pour comparer le comportement des sites web dans des environnements normaux versus restreints afin d'identifier les divergences qui indiquent la pertinence de certains attributs UBE. Ce cadre permet la restriction sûre des informations UBE non essentielles, améliorant ainsi le respect de la vie privée de l'utilisateur.

2. Nous utilisons Similarity Radar pour étudier l'impact de l'en-tête HTTP User-Agent (UA) sur le web. De manière similaire à la comparaison entre l'UBE standard et restreint décrite précédemment, nous explorons 270 048 pages web sur 11 252 domaines en utilisant trois navigateurs standard et un ensemble de navigateurs « None » — des clones des navigateurs standard avec leur UA et informations associées définies comme « None ». Nous comparons ensuite la similarité entre les navigateurs standard et les navigateurs « None ». Avant l'exécution de JavaScript (JS), nous observons une similarité de 100 %, indiquant que les en-têtes UA sont aujourd'hui sans pertinence pour la fonctionnalité des sites web. Cependant, après l'exécution du JS, 8,4 % des pages web explorées changent. Ces changements sont principalement dus à des scripts tiers liés à la publicité, à la détection de robots et aux réseaux de diffusion de contenu. En approfondissant, nous classons ces changements selon des niveaux de gravité en termes d'utilisabilité, la plupart étant

de légères modifications CSS et un plus petit pourcentage causant des problèmes d'utilisabilité significatifs. Cela indique l'obsolescence de l'en-tête UA malgré leur importance dans le fingerprinting des navigateurs.

3. Nous examinons l'usage des bannières de consentement aux cookies sur différents continents, en nous concentrant sur la manière dont la visibilité et l'impact varient selon la localisation géographique. Nous utilisons une nouvelle technique de détection visuelle automatisée basée sur Similarity Radar pour détecter efficacement les bannières de consentement. Pour chacun des cinq pays—Brésil, France, Japon, Afrique du Sud et États-Unis—nous avons analysé 14 078 sites web, en répétant ce processus trois fois sous différents scénarios d'interaction : d'abord sans interagir avec les bannières de consentement aux cookies, puis en les acceptant, et enfin en les refusant. Les résultats révèlent des disparités dans la prévalence des bannières de consentement, la France affichant la plus haute visibilité à 69 % et le Japon la plus basse à 27 %. Nous observons également que les cookies tiers augmentent après que les utilisateurs interagissent avec les bannières de consentement, en particulier aux États-Unis, tandis que la France maintient le nombre le plus bas de cookies tiers en raison de sa réglementation stricte. Cela indique une corrélation entre la visibilité des bannières de consentement et les pratiques de traçage des utilisateurs.

Acknowledgements

My PhD journey began during the COVID-19 pandemic amidst lockdowns in both France and Rwanda, where I was residing. Navigating this challenging period required the collective support of many people. First and foremost, I extend my deepest gratitude to my supervisors, Romain Rouvoy and Pierre Laperdrix. Their trust in my abilities to work with them, along with their support, patience, and guidance, has been invaluable to me. Romain, I am profoundly thankful to you and your family for assisting me in settling into Lille. Your help made what could have been a daunting transition smooth and welcoming. Pierre, I am grateful for your active support and prompt feedback in our shared office environment.

I am also grateful to Imane Fouad, whose collaboration has been instrumental to my PhD experience. Imane, your timely feedback and thoughtful discussions have enriched my work. Additionally, I thank the members of the Spirals team, whose camaraderie has made my past three years enjoyable. Lionel Seinturier, thank you for leading Spirals with excellence and for creating an environment of strong work ethic and lasting memories. To Adrien, Rémy, Daniel, Pierre J., Naif, Salman, Antoine, Brell, Hugo, Maxime, Maryam, Sihem, Belkis, Iliana, Emile, Thibaut, Edouard, Olga, Walter, Clément, Simon, and all the members of Spirals: your conversations during seminars, coffee breaks, lunches, corridor encounters, pots, afterworks, home visits, and countless other interactions have made this journey unforgettable.

Moreover, the completion of this journey would not have been possible without the unwavering love and support of my family. Transitioning our lives to a different country during a global pandemic was a formidable challenge, one that was made manageable thanks to the resilience and encouragement of my wife — Christella, and children — Khaliq, Cleo, and Elon. Christella, your steadfast support enabled me to continue my work despite the upheavals we faced. I am thankful to my parents — Origène Rutayisire & Béatrice Mukanyandwi, and my sisters — Emelyne Ingabire Niwe, Ines Michelle Iradukunda, Teta Sonia Kanakabakwiye, and Reine Axcelle Nancy Kanyange — whose constant encouragement has been my anchor. I also extend my heartfelt thanks to our friends Christian & Megan Hardin, Daniel & Liz Koenigsberg, and every other friend in Lille and elsewhere. Thank you all!

Jean Luc INTUMWAYASE
Lille, December 2024

Table of contents

List of figures xiii

List of tables xvii

1 Introduction 1

1.1 Motivation . . . 1
1.2 Contributions . . . 2
1.2.1 Determining information relevance in the user's browsing environment 2
1.2.2 Exploring the impact of device information on the web . . . 2
1.2.3 Exploring the geolocation impact on enforcing privacy policies . . 3
1.3 List of Publications . . . 3
1.4 List of Tools . . . 4
1.5 Outline . . . 4

2 Background & Related work 7

2.1 Background . . . 7
2.1.1 How the web works . . . 7
2.1.2 Misuse of the user's browsing environment . . . 9
2.2 Measuring web similarity . . . 10
2.2.1 Analyzing text content of a web page . . . 10
2.2.2 Analyzing the HTML structure of a web page . . . 10
2.2.3 Analyzing JavaScript & CSS . . . 12
2.2.4 Analyzing visual rendering of a web page . . . 13
2.2.5 Detecting web page breakage . . . 15
2.3 Minimizing browser distinctiveness . . . 17
2.3.1 Analyzing leaks from the user's browsing environment . . . 17
2.3.2 Restricting access to the user's browsing environment . . . 18
2.4 The geolocation impact on privacy policies . . . 21
2.4.1 Identifying cookie consent notices . . . 21
2.4.2 User interaction with cookie consent banners . . . 25

3 Determining information relevance in the user's browsing environment 29
3.1 Overview . . . 29
3.2 What is UBE information? . . . 30
3.3 Simulating access to UBE . . . 32
3.3.1 Simulating geolocation . . . 32
3.3.2 Simulating device and browser information . . . 34
3.4 Designing the crawler . . . 37
3.4.1 Crawl orchestration . . . 38
3.4.2 Dealing with dynamicity of web pages . . . 39
3.5 Similarity Radar . . . 42
3.5.1 Computing similarity scores for each dimension . . . 43
3.5.2 Computing the similarity score S_{score} . . . 46
3.5.3 Determining information relevance in UBE . . . 46
3.6 Threats to validity . . . 47
3.6.1 Bot detection . . . 47
3.6.2 Investigating web page functionality . . . 48
3.6.3 Scope of relevance of UBE attributes . . . 48
3.7 Conclusion . . . 49

4 Exploring the impact of device information on the web 51
4.1 Overview . . . 51
4.2 Motivation . . . 52
4.3 UA-Radar: Measuring Web Similarity in the Wild . . . 53
4.3.1 Overview . . . 53
4.3.2 Implementation Details . . . 55
4.4 Exploring the Impact of UA Changes . . . 58
4.4.1 Crawl Description & Statistics . . . 59
4.4.2 Empirical Results & Findings . . . 60
4.5 Discussion . . . 75
4.6 Impact of None-browsers on web privacy . . . 75
4.7 Threats to Validity . . . 76
4.8 Conclusion . . . 76

5 Exploring the geolocation impact on enforcing privacy policies 79
5.1 Overview . . . 79
5.2 Motivation . . . 79
5.3 Web Crawling Methodology . . . 81
5.3.1 Preparing the crawler . . . 83
5.3.2 Connecting to countries . . . 84
5.3.3 Visual detection of banners . . . 85
5.3.4 Classification of banners . . . 87

5.4 Analysis . . . 88
5.4.1 Data collection . . . 88
5.4.2 Prevalence of cookie consent banners . . . 89
5.4.3 State of cookies . . . 90
5.4.4 State of tracking . . . 93
5.5 Discussion . . . 96
5.6 Threats to validity . . . 97
5.7 Conclusion . . . 99

6 Conclusion **101**
6.1 Contributions . . . 102
6.1.1 Determining information relevance in the user's browsing environment 102
6.1.2 Exploring the impact of device information on the web . . . 103
6.1.3 Exploring the geolocation impact on enforcing privacy policies . . . 103
6.2 Future work . . . 104
6.2.1 Short-term perspectives . . . 104
6.2.2 Long-term perspectives . . . 106
6.3 Concluding note . . . 106

Bibliography **128**

A Appendices **129**

Appendices **129**
A Exploring the impact of device information on the web . . . 129
A.1 Crawled Domains . . . 129
A.2 Navigator Properties Exposed During the Crawl . . . 129
A.3 UA-Radar . . . 130
A.4 Running the Application . . . 130
A.5 Changing Test Files . . . 131
A.6 HTML Content Comparison . . . 134
B Exploring the geolocation impact on enforcing privacy policies . . . 135
B.1 Crawler Details . . . 135
B.2 Results of Our Experimentation . . . 135

List of figures

2.1 Detecting web page structure from HTML tags, heuristics, and visual cues. This approach [26] mimicks the user's visual perception of the web page (left image) and converts it into web structures (right image). 11

2.2 Different JavaScript syntax for the same behavior across two sets of examples. The first set demonstrates two ways to define a function that adds two numbers, and the second set shows two approaches to fetch data from a URL and parse it as JSON. When measuring the similarity of the two sets, an ideal approach should consider that both the left and right sides in the above sets are similar. 13

2.3 Detection of visual bugs in an HTML5 canvas game, as proposed by Macklon *et al.*, involves identifying any mismatch between an expected image and an actual image. If there is a difference between the expected image `(a)` and the actual image `(b)`, that difference indicates the presence of a visual bug in the game. This is illustrated in image `(c)`. This approach relies on an arbitrary representation of objects and image comparison. . . 16

3.1 Categories of UBE information: Geolocation, Device, and Browser Information 30

3.2 Browser geolocation exposing the user's coordinates due to bypassed permission-granting process in Puppeteer. 33

3.3 Comparing real devices with Puppeteer's UBE simulation while visiting `https://www.whatsapp.com/download/`: (a) real MacBook Pro, (b) simulated MacBook Pro, (c) real iPhone 13, (d) simulated iPhone 13, (e) simulated Galaxy S8. 37

3.4 41

3.5 42

3.6 42

3.7 Similarity score S_{score} for two web pages $(v'_{ij}(t), v'_{ik}(t))$ 43

4.1 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. 54

4.2 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. 55

4.3 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. 57

4.5 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. 66

4.6 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. 70

4.7 Example of "unusable" problem severity: access to the web page is intentionally restricted when using a None Browser. 72

4.8 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. 74

5.1 Overview of our methodology for analyzing the prevalence of cookie consent banners and the state of cookies and tracking. The process begins with the preparation of the crawler, ensuring accurate data collection and integration of browser extensions to accept or reject cookie consent banners. This is followed by establishing a geo-location connection using `HOXX VPN`. The crawler is then executed in three distinct instances: without interaction with the banner, with `IDCAC` accepting consent banners, and with `Consent-o-Matic` rejecting them. Screenshots are captured during the first instance for subsequent visual detection of banners. The final stages involve identifying websites displaying the banners and analyzing the state of cookies and tracking. 81

5.2 A comparison of the success rate of interaction, through either acceptance or rejection of cookies, by different Firefox browser extensions. The results are based on a sample experiment conducted on 100 domains, aiming to decide on suitable extensions for our analysis of the state of cookies and tracking, which required not removing cookies or blocking HTTP requests. 82

5.3 Distribution of third-party cookies across websites for different countries. The upper plot illustrates the number of third-party cookies served to the user when no banner is shown and the plot at the bottom illustrates when the user has not interacted with the consent banner yet. 85

5.4 Distribution of third-party cookies across websites for different countries. The upper plot illustrates the number of third-party cookies after the user accepts cookies, and the plot at the bottom illustrates the situation when the user rejects consent to cookies. 86

5.5 Comparative analysis of cookies set during the crawl across five countries under the `No Banner` and `No Interaaction` scenarios. 91

5.6 Comparative analysis of cookies set during the crawl across five countries under the `Accept` and `Reject` scenarios. 92

5.7 Prevalence of consent banners, ID cookies, sites serving ID cookies, third-party requests, and tracking requests in different scenarios: the upper figure shows the `No Banner` scenario, when no consent banner is shown in countries other than FR and consequently no consent prevalence is measured. The figure at the bottom shows the `No Interaction` scenario when a consent banner is shown but the user has not yet interacted with it. 94

5.8 ID cookies, sites serving ID cookies, third-party requests, and tracking requests in different scenarios: the upper figure shows the `Accept` scenario while the figure at the bottom shows the `Reject` scenario. 95

5.9 Comparative analysis of HTTP requests made during the crawl across five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US). 97

5.10 Comparative analysis of HTTP requests made during the crawl across five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US). 98

List of tables

4.1 Summary of crawled resources 60
4.2 Internet categories of the UA-dependent domains 67
4.3 Summary of problem severity levels for UA-dependent domains 67
4.4 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. 71
4.5 Web privacy implications of UA usage: this table presents an analysis of domains based on their interaction with UA. 76

5.1 Comparison of the number of domains displaying cookie consent banners and their corresponding prevalence percentages in five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US), as identified by two different detection methods - our automated visual approach and the `z-index` approach. 83

A.1 Categories of Crawled Domains 130
A.2 Navigator Properties for Chromium Browsers 131
A.3 Navigator Properties for Firefox Browsers 132
A.4 Navigator Properties for WebKit Browsers 133
A.5 Results for No Interaction 136
A.6 Results for Accept 136
A.7 Results for Reject 136

Chapter 1

Introduction

1.1 Motivation

The web supports a broad diversity of devices and configurations. Anyone with a browser-enabled computing device and an Internet connection can access it from anywhere. So, websites rely on information like whether the user's device is a phone or a computer, operating system, browser, and geolocation to adjust layouts on each user's device. This information, dubbed the "user's browsing environment", is key to web functionality but can also be problematic when misused [61, 142, 24]. For example, knowing that a user is accessing a website from a specific location enables the website to use the nearest web server to load its content faster. However, websites also store location information not just for faster loading speeds, but for finding patterns in user behavior to influence their preference for products [11, 31, 106]. This misuse of the user's browsing environment constitutes the "web privacy" issue pertaining to the lack of transparency in the collection, treatment, and sharing of user data [53].

The issue of web privacy has had a profound impact on the evolution of web technologies. On one hand, protective measures have been implemented to prevent websites from misusing the user's browsing environment [69, 221, 226, 156, 81]. On the other hand, websites have developed methods to bypass these protective measures [67, 89]. Governments have also stepped in, introducing web privacy regulations that require websites to be transparent about their use of data [84, 27, 22]. These developments have influenced the evolution of web technologies. However, one overlooked factor is the necessity of information accessible through the user's browsing environment. No work has examined whether this information is essential to the functionality of the web today, given the evolution of its technologies. Instead, the focus has been on protecting against misuse of the environment and requiring transparency on data practices.

Our main objective in this thesis is to better understand the relevance of the information exposed in the user's browsing environment, given their role in bypassing privacy protective measures. Specifically, we examine the impact of exposing device information on the web

today. Additionally, we explore the impact of geolocation on web privacy, given that regulations are neither omnipresent nor enforced the same way. To address these points, we pose and answer the following research questions:

- How can we investigate the relevance of information in the user's browsing environment?
- What is the impact of device information on today's web? Can web technologies support an unknown device?
- Does geolocation have an impact on the enforcement of privacy regulations?

1.2 Contributions

1.2.1 Determining information relevance in the user's browsing environment

The primary contribution of this thesis is the development of a tool to compare web pages across various similarity dimensions. This helps us understand the impact of changes in the user's browsing environment, which in turn indicates the relevance of the information being investigated. We instrumentalize multiple visits of the same web page with different settings, then measure the similarity of the HTML structure, text content, JavaScript, CSS, visual rendering, and the state of cookies and tracking each time we visit the web page. We compute a similarity score on each of those measured dimensions, and these scores help understand the impact of changes in the user's browsing environment on the visited web page's functionality. High similarity scores indicate the irrelevance of the investigated settings. This detailed analysis at scale enhances our understanding of the relevance of exposed information in the user's browsing environment, ultimately contributing to the improvement of web privacy.

1.2.2 Exploring the impact of device information on the web

In the early days of the web, visiting the same web page from different browsing environments could provide very different results. Due to different rendering engines behind every browser, some elements of a web page could break or be positioned in the wrong location. At that time, the *User Agent* (UA) string was introduced to expose device information for content negotiation. UA provides information such as whether the browsing environment is on a desktop or mobile, the operating system, type and specific version of the used browser. By knowing this information, a developer could provide a web page that was tailored for that specific browsing environment to remove any usability problems. Over the past three decades, device information remained exposed by UA, but its necessity is debated. Browsers now adopt the exact same standards and use the same languages to display the

same content to user. Moreover, the diversity of device types, operating systems, and browsers has become so large that UA is one of the top contributors to tracking users in the field of browser fingerprinting. Our goal is to understand the impact of exposing device information through UA and the handling of unknown UA on the web. We use our web page similarity approach to compare web pages visited using known UA versus unknown UA. We crawl 270, 048 web pages from 11, 252 domains using 3 different browsing environments to observe that 100% of the web pages were similar before any JavaScript was executed, demonstrating the absence of differential serving. Our experiments show that only a very small number of websites are affected by unkown device 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 stop exposing device information in the user's browsing environment.

1.2.3 Exploring the geolocation impact on enforcing privacy policies

Regulatory frameworks, such as GDPR in Europe, CCPA in the United States, and LGPD in Brazil have made cookie consent banners a standard feature on websites, seeking user consent for cookies and tracking technologies. However, the visibility and impact of these banners may vary based on the user's geographical location. We introduce a novel automated visual detection technique to explore the enforcement of cookie consent banners. This methodology outperformes earlier techniques that depend on manual observations or inspection of HTML/CSS elements. Our analysis of 70, 390 web pages visited across five countries on different continents reveals geographical disparities in banner visibility, with France exhibiting the highest prevalence at 69% and Japan the lowest at 27%. This variation in visibility correlates with the state of cookies and tracking. Our further analysis of 351, 950 web pages indicates that third-party cookies increase post-banner interactions, notably in the US. Conversely, France, with its stringent regulations, maintains the lowest number of third-party cookies. Tracking trends follow these patterns with a relationship between banner visibility and user tracking. We demonstrate the importance of visually detecting banner visibility, as it directly impacts tracking activities and provides insights into user consent practices.

1.3 List of Publications

[112] Jean Luc Intumwayase, Imane Fouad, Pierre Laperdrix, and Romain Rouvoy. "UA-Radar: Exploring the Impact of User Agents on the Web". In: *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*. 2023, pp. 31–43

1.4 List of Tools

- UA-Radar: Web Similarity Framework implemented in Node.js [111].
- Cookie Consent: Crawler, code for automatic visual detection, and extensions to interact with cookie consent notices [110].

1.5 Outline

In this chapter, we introduce the impact of information exposed in the user's browsing environment on web functionality and privacy. We present our motivations for the thesis, highlighting the use of device information and geolocation to optimize user experience, but also to track user behavior, raising privacy concerns. We introduce our contributions, including the development of a tool to compare web pages across various dimensions, exploring the impact of device information on the web via the UA string, and exploring the geolocation impact on enforcing privacy regulations. These contributions aim to enhance our understanding of the user's browsing environment and inform the development of better protective measures for web privacy.

In Chapter 2, we define the user's browsing environment and explain how it works. We review previous work on measuring web similarity. We present the contributions and gaps of existing studies on the impact of device information on the web. We do a similar presentation on the impact of geolocation on enforcing privacy regulations on the web. We highlight the common or different aspects those studies present to our approaches in this thesis.

In chapter 3, we introduce dimensions of similarity in a web page, such as the HTML structure, text content, JavaScript, CSS, visual rendering, and the state of cookies and tracking. We use these dimensions to present a methodology for investigating the impact of changes in the user's browsing environment on a web page. The result is a tool for measuring the impact of changes and enabling analysis to determine the relevance of information in the user's browsing environment. We conclude the chapter by discussing the scope of our web similarity tool.

In Chapter 4, we use our web similarity tool to examine the relevance of device information on the web through the use of UA. Can unknown UA be supported on the web? We compare websites' treatment of known UAs and 'None-browsers'. We further evaluate the changes caused by 'None-browsers' and their impact on the usability of the visited web pages. We discuss our findings, and conclude with thoughts on the future of UAs and device information in the user's browsing environment.

In Chapter 5, we introduce a methodology for automatic visual detection of cookie consent banners. This helps us analyze their prevalence and understand countries where enforcement of privacy regulations is weak or nonexistent. We further use our web similarity tool to examine the state of cookies and tracking on the websites when visited

from different countries with different privacy regulation landscape. We conduct cookie and tracking analyses for different scenarios of banner interactions, including non-interaction, acceptance, and rejection of the cookie consent banners.

We conclude this thesis with Chapter 6, reflecting on our contributions and their implications to web privacy. We discuss our short and long-term perspectives and ways to further our contributions. From enhancing the similarity metrics to uniformity of shareable information, we present our views to further the work in this thesis and enhancing web privacy.

Chapter 2

Background & Related work

In this chapter, we provide background for concepts in this thesis such as the 'user's browsing environment', 'web page similarity', and 'web page breakage'. First, we explain how the web works, with an emphasis on how the evolution of its technologies led to the aforementioned issue of web privacy. We then present related work and discuss the methodologies used in those studies, relevant to our contributions in this thesis. We explain the concept of web similarity and review previous research on similarity dimensions in a web page and the measurements used to quantify the similarity of two web pages. Furthermore, we discuss web page breakage, including key concepts and existing research on detecting and addressing breakage. Next, we delve into the topic of device information and its impact on web functionality. We explain key concepts related to device information, such as the UA string, and review previous work on minimizing the exposure of device information and its impact on the web. We also explore existing approaches to detect user consent in light of privacy regulations. Additionally, we review previous research on user tracking in different geographical locations to address disparity in tracking. Finally, we conclude with a discussion of the methodologies used in this thesis to address the topics covered.

2.1 Background

2.1.1 How the web works

In 1989, Tim Berners-Lee proposed merging technologies of computers, data networks, and hypertext into a global information-sharing system [2]. This system came to be known as the World Wide Web, or just the web. To access information on the web, a user needs a computer, a browser, and an Internet connection. Web servers provide information on the web. A web server is a system that accepts user requests from the browser, also called a 'client', and sends back responses. A set of protocols is used for the user's browser to communicate with a web server. TCP/IP is used to establish a connection between the computer running the browser and the computer hosting the web server. The Internet

Protocol (IP) assigns a unique address to each device connected to the Internet, ensuring connection between the correct devices when a user's browser communicates with a web server [189]. The Transmission Control Protocol (TCP) provides reliable delivery of information between connected devices [190].

Once the computer running the user's browser and the computer hosting the web server are connected, the next step is to establish a 'client-server' communication. This involves setting up communication between the browser and the web server. Hypertext Transfer Protocol (HTTP) is used to create that communication [163]. The browser sends an HTTP request to the web server, requesting for a resource such as a hypertext document, and the web server responds to the request with an HTTP response. The HTTP response often contains the requested resource in the form of HTML code. Hypertext Markup Language (HTML) is a standard annotation of information on the web, used to create hypertext documents [16]. Information on the web is accessed through hypermedia documents, which extend hypertext by incorporating non-textual elements such as images, audio, video, and interactive elements. These documents are commonly referred to as HTML documents. In this thesis, we use the term 'HTML document' to refer to a hypermedia document.

An HTML document is written following a structure called Document Object Model (DOM) [58], which represents the document as a tree of nodes with relationships to each other. That DOM representation makes it possible for other programming languages to interact with the HTML document, and modify its structure or content. That is why it can include separate files called Cascading Style Sheets (CSS) documents also called stylesheets [49], used to specify the styling of information in the document. An HTML document can also include separate files called JavaScript [63], or simply scripts, which are programs executed by browsers to add interactivity to the website. So, when a user visits a website to access information, their browser makes an HTTP request to the web server hosting that website, and the web server responds with an HTML document. The HTML document that the web server responds with is then loaded in the browser, which can also load separate files including stylesheets and scripts if specified by the HTML document. When the browser has completed loading all files as required by the HTML document, the final presentation of that HTML document in the browser is called a 'web page'. The process of presenting the web page in the browser is called 'web page rendering'.

The user's browsing environment consists of the context in which a web page is rendered. This includes data on the application layer such as device information, operating system, user's geolocation, the browser and its configurations. By the late 1990s, millions of users were already accessing the web using various browsing environments [2]. At that time, web page rendering was different in every browser as vendors pursued market domination. This created accessibility issues, as a website could render correctly in one browser and not in another. Messages reading "best viewed in [specific name of browser]" became commonly displayed on websites as developers with limited resources chose to focus on specific browsers. Then the World Wide Web Consortium (W3C), a consortium of organizations

in charge of the web, developed standards to promote consistent web page rendering across different browsers regardless of their underlying engines.

2.1.2 Misuse of the user's browsing environment

Some web technologies have become a double-edged sword by providing functionality while being detrimental to user privacy. Web cookies, for example, are small files that store information about user activities on a website. Cookies were invented for useful purposes such as knowing the user session and keeping them logged in, their location and language, or storing items in the cart while the user is purchasing products on the web. However, cookies are also used to keep track of user activities even when the user is visiting other websites [13]. For example, some cookies are set by websites other than the one being visited by the user. These are called third-party cookies and are used to identify users and keep track of their activities as they visit different websites.

Similarly, JavaScript is a programming language that was invented to give developers the capacity to modify web pages once they were loaded in the browser, but it came with unintended consequences [14]. In the early days of the web, web pages were 'static', meaning that browsers rendered web pages as exactly stored on the web servers. To modify a static web page, the website administrator had to modify the content of that page on the web server and the user had to make a new request of the web page to see the changes in the browser. This process was cumbersome to website administrators and users alike as each was required to take action to see modified content. Also, the functionality of the web was limited to websites showing content to users, and users could not generate or send content to web servers.

JavaScript resolved those challenges: web pages could be modified without the need for users to make new requests, and users could generate content and send it to web servers [82]. Web pages with those capabilities were called 'dynamic web pages'. A dynamic web page, as opposed to a static web page, is constructed during the rendering of the web page. JavaScript also revolutionized browser graphics, animations, games, data streaming, and extended the functionalities of the web [47, 56, 248, 138]. Most common web functionalities are written, tested, and packaged in JavaScript libraries, which authors often share for free. Many package managers, which automate the distribution of JavaScript libraries are also available on the web for free [136, 42, 201]. To promote efficient use of JavaScript on the web, many frameworks emerged with distinct purposes proposing standard ways to write code and reuse it [94, 4, 76, 183]. As a result, because of its many applications, JavaScript became so popular that 99% of websites use it at present [225]. Every web browser offers a JavaScript engine to execute scripts on the user's device.

However, JavaScript is also often exploited to collect and share user data without their knowledge [137, 154, 242, 64]. Many studies found JavaScript to be an enabler for websites

to extract data from the user's browser environment [205, 253, 141, 65, 131]. JavaScript is also a channel for user tracking. 'User tracking' are techniques used to identify users and to keep track of their activities on the web. These techniques may or may not require the user's consent. User tracking without consent occurs because users do not consent to such practices when asked with transparency [126].

2.2 Measuring web similarity

Web similarity is the degree of resemblance between two or more websites in terms of similarity dimensions such as their structure, content, and visual rendering among others. To the best of our knowledge, no previous research explored dimensions of web similarity to the end of detecting web page breakage. Previous studies have addressed web similarity or web page comparison for different purposes, with methodologies worth reviewing and emulating in case they fit our purpose. In this section, we will discuss works relevant to our approach while highlighting differences or commonalities.

2.2.1 Analyzing text content of a web page

Previous studies analyzing the text content of a web page mostly devise the best strategy for detecting when the content has been updated [34, 219, 146, 70]. From inspecting HTTP headers such as `Last-Modified`, `Date`, and `Expires` to detecting changes in the size of web pages or measuring differences in word count, those approaches give no insight into the actual changes of content inside the HTML tags. To gain more insights into content change, some studies introduced methods to classify web pages depending on their content [212, 100, 98, 88]. Tombros *et al.* contributed to the classification techniques categories based on word significance and their frequency distribution [219]. While this may have been insightful, it still did not solve the issue of measuring changes in the actual text.

Detecting changes in text content requires tracking text alterations inside each visible HTML tag on both web pages. This is challenging due to large volumes of text on web pages, which is computationally intensive. So, simply comparing text content in HTML tags is inefficient and does not scale for large web pages. Our goal is to develop a comparison method that is both efficient in tracking all changes despite the size of the web page. To achieve this, we adopt a method for disregarding similarities and focusing on actual changes to ensure that the process remains computationally feasible. We present this approach in detail in Chapter 3.

2.2.2 Analyzing the HTML structure of a web page

Early studies used heuristics to deduce web page structure from a combination of HTML tags, semantic blocks, and visual cues [36, 26]. The methods analyzed HTML structure

and adapted it based on detected patterns using heuristics. These approaches aim to detect the structure of a web page based on how users perceive it visually rather than following the DOM hierarchy. Figure 2.1 illustrates how the approaches deduce HTML structure. While mimicking the user's perception can help classifying the importance of changes on a web page, these approaches may overlook changes of lesser significance. We were interested in detecting any alterations to the structure of the web page for our analysis of the relevance of information within the user's browsing environment. What may seem less important from the user's perspective could potentially be relevant to the web page's usability, for instance. Furthermore, it is difficult to keep up with design choices on the web, and without that understanding these approaches become misleading.

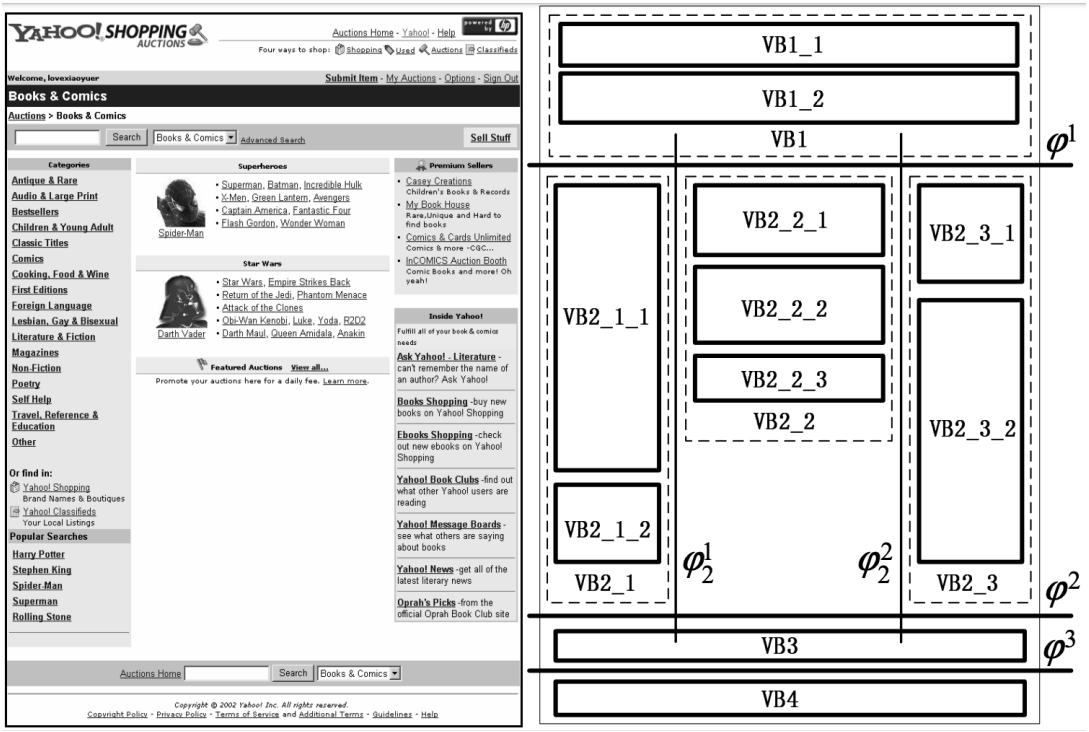


Figure 2.1: Detecting web page structure from HTML tags, heuristics, and visual cues. This approach [26] mimicks the user's visual perception of the web page (left image) and converts it into web structures (right image).

Choudhary *et al.* solve the previous subjective DOM interpretation indirectly [37]. The authors introduce WEBDIFF to automate the detection of discrepancies in a web application's behavior and appearance when accessed through various browsers. WEBDIFF's approach to comparing the DOM structure combines HTML tags and their XPath to compute a 'match index' for pairs of DOM nodes of different web pages to determine how closely those nodes match. After matching the DOM nodes, WEBDIFF compares them visually using the Earth Mover's Distance (EMD) metric [192]. The effectiveness of this approach in identifying differences in the structure and visual rendering of web pages could be useful in detecting web page breakage. However, its stated limitation in

handling embedded objects could lead to false positives, suggesting change where content has simply not changed.

Other studies focused on Tree Edit Distance (TED) [217] and Flexible Tree Matching (FTM) [128, 129] to analyze web structure with a focus on DOM. TED and FTM are both used to measure the similarity between two tree structures. TED's approach uses the smallest number of changes (edits) to transform one tree into another, while FTM looks for a 'best match' between nodes of two trees. TED is strict about the positioning of the nodes in the tree while FTM is less strict. That flexibility makes FTM able to compare large web pages in realistic times although computationally intensive. Brisset *et al.* proposed Similarity-based Flexible Tree Matching (SFTM) to address that computational intensity [23]. SFTM improves FTM by adding similarity labels and propagation. This gives TED-like quality while offering improvement in computation times. This makes SFTM our choice in measuring the similarity of HTML structure as described in Chapter 3.

2.2.3 Analyzing JavaScript & CSS

The methods to analyze JavaScript are usually applicable to CSS due to their shared characteristic of manipulating of the DOM. Furthermore, there are common technical concepts behind JavaScript and CSS, as well as other general-purpose languages, such as syntax and control structures. Therefore, our review also includes methodologies for source code similarity in languages beyond just JavaScript and CSS.

Due to the size of the code base, previous studies resorted to techniques to measure code pair and structure similarity [256, 92], or to abstracting the code to measure similarity more effectively [123, 155, 99, 172]. The problem with those approaches in our context is the level of granularity in their measurements. When we compare JavaScript on a web page, we want to understand any change however minor it may be. The limitation of the previously mentioned approaches is that the abstraction process can conceal important details relevant to usability of the web page. On the other hand, not every difference in lines of code is worth measuring. For example, there may be a different syntax for the same code block as illustrated by listings in figure 2.2. If that difference does not change the behavior of the code, it is not worth tracking for measurements. The same goes for comments or other changes in the code file that are not interpreted differently by JavaScript or CSS.

```
function add(a, b) {
  return a + b;
}
```

```
const add = (a, b) => a + b;
```

```
function fetchData() {
    return fetch('url').then(
        response => response.json())
        ;
}
```

```
async function fetchData() {
    const response = await fetch('
        url');
    return response.json();
}
```

Figure 2.2: Different JavaScript syntax for the same behavior across two sets of examples. The first set demonstrates two ways to define a function that adds two numbers, and the second set shows two approaches to fetch data from a URL and parse it as JSON. When measuring the similarity of the two sets, an ideal approach should consider that both the left and right sides in the above sets are similar.

Dotzler *et al.* presented an approach to convert JavaScript and CSS code into an Abstract Syntax Tree (AST) to analyze and compare code at a granular level [59]. The study introduced optimized algorithms for 'tree diffing' such as identifying subtrees and mapping node movements to enhance detection of minor code changes. In the context of this thesis, this approach could be applied in comparing the JavaScript and CSS code of two web pages to measure their similarity. This methodology was later added in GumTree, a syntax-aware diffing tool, to detect specific changes that lead to issues in JavaScript functions or CSS rules [73]. This allows for effective analysis of JavaScript and CSS code without computational constraints. Methods based on Abstract Syntax Tree (AST) provide a better understanding of code similarity beyond textual comparison [17, 249, 143]. Other studies focused on approaches to summarize code in their similarity analyses [243, 34, 121].

On GumTree, Falleri *et al.* addressed the challenges of computing detailed and accurate edit scripts for change in source code files [73]. Their methodology operates at the level of AST granularity enabling the representation of the structure of the code more accurately by including move actions alongside the usual add, delete, and update actions. This is essential in representing code editing patterns such as refactoring where code is moved but not altered. The primary objective of the algorithm was not to just find the shortest sequence of actions between two versions of a file but to identify a sequence that closely reflects the actual changes. We used GumTree for computing the similarity of JavaScript and CSS as discussed in Chapter 3.

2.2.4 Analyzing visual rendering of a web page

Previous studies relied on color and text features in measuring the visual similarity of web pages in what in the context of web page similarity can seem like an oversimplification of

the visual content of a web page. Varish *et al.* developed a Content-Based Image Retrieval (CIBR) system that focuses on retrieving similar images from a database based on visual features [231]. Their methodology was to convert RGB images to HSV (Hue, Saturation, Value) format and using quantized histograms of the Hue and Saturation components. Color features are extracted from these histograms to indicate the distribution of the colors in the images. When considering the application of CIBR to comparing visual rendering of web pages, relying on the color and texture of features for analyzing elements such as text, icons, and layout structures is inefficient. This methodology assumes that images share consistent patterns as in a collection of photographs. Unlike a typical photograph, screenshots of web pages contain complex details that affect the appearance of a web page including formatting options such as font size and spacing. Therefore a more comprehensive approach is needed to detect any changes in visual rendering.

Wenyin *et al.* introduced an approach to identify phishing websites by comparing their visual similarities with legitimate web pages [250]. Their methodology focused on three metrics to measure visual similarity: block level similarity, layout similarity, and overall style similarity. These metrics were computed based on a detailed segmentation of web pages into salient blocks. Block level similarity was quantified as the weighted average of similarities between matched block pairs, focusing on features like color for text blocks and dominant color for image blocks. Layout similarity was measured by comparing the structure and arrangement of those blocks. The overall style similarity was evaluated based on the style feature histograms of the web pages, considering elements like font, background, and text alignment. Similar to Varish *et al.*, while innovative, this approach is somewhat simplistic due to the level of details that could change in a block. For example, changes such as formatting or broken elements in a block may be overlooked because of their granularity.

Law *et al.* worked on web page similarity using a method that combines visual and structural analysis [139]. They processed screenshots of visited web pages to extract color and edge-based features in order to identify changes between web page versions. Special attention was given to the visible part of web pages, considering that significant information is usually located in the parts visible without scrolling. The study also focused on extracting structural features from the web page's source code. This included analyzing hyperlinks, image URLs, and operations detected between VIPS structures [26] of different versions of a web page. Those features helped in understanding the structural changes that may not be visually apparent but are essential in determining the similarity between web page versions. While innovative in combining visual and structural analysis, this approach falls short in efficiently detecting changes in visual rendering. This is due to the assumption that the most significant information is located in the parts visible without scrolling, and the focus on detecting structural changes rather than changes in visual rendering.

Hashmi *et al.* introduced QLUE, an approach that uses computer vision techniques to emulate human perception in evaluating web page similarities [101]. QLUE overcame the subjectivity of human-based evaluations by breaking down web pages into components, matching those components between the original and modified web pages, and then calculating a similarity score. QLUE marked an advancement in web page evaluation and opened avenues for future research including refining the tool to handle dynamic content, interactive elements, and integrating it with other web measurement tools. However, QLUE's capacity to process a web page in minutes is impractical for large-scale studies. To that end, we introduce our novel approach to measuring visual rendering. We discuss that in Chapter 3.

2.2.5 Detecting web page breakage

One of the observed impacts of changes in a user's browsing environment is 'web page breakage'. This refers to any unintended consequence that occurs to a web page due to changes in the user's browsing environment. Known cases of web page breakage include blank pages, malfunctioning elements such as forms, links, buttons, slowdowns, freezing, crashing, and other errors and display issues. There is currently no standard definition for web page breakage, and there is no comprehensive dataset of known cases. While there are mechanisms to collect user reports of breakage, these have not led to a systematic definition of the term [165]. As a result, the ambiguity surrounding the term 'web page breakage' makes it difficult to automate its detection. In this thesis, we address this challenge by resorting to measuring the similarity between different versions of a web page and defining any significant dissimilarity as web page breakage. We discuss more on our approach in Chapter 4. Existing methodologies for detecting web page breakage can be categorized into two main approaches based on their scope of analysis: one focuses on the analysis of individual web elements within a web page, while the other analyzes the web page as a whole.

One example of analyzing individual web elements is the framework developed by Fouquet *et al.*, which inspects web page features such as images, forms, and buttons when JavaScript is blocked [79]. The framework classifies these web elements as either working or broken based on the DOM state after the initial page load with JavaScript blocked. However, a limitation of this approach is that it is difficult to leverage in the wild because it requires adjusting the detection heuristics by thinking of every possible breakage scenario. This approach could be effective if either of the following conditions are met: first if there was a comprehensive dataset of instances of web page breakage; second, if there was a universally accepted definition of what constitutes web page breakage. Unfortunately, as previously mentioned none of those conditions are met. Additionally, the focus on individual web elements instead of the whole page makes it challenging to determine the cause of the breakage in a web page.

(a) Expected image (b) Actual image (c) Bug detection

Figure 2.3: Detection of visual bugs in an HTML5 canvas game, as proposed by Macklon *et al.*, involves identifying any mismatch between an expected image and an actual image. If there is a difference between the expected image `(a)` and the actual image `(b)`, that difference indicates the presence of a visual bug in the game. This is illustrated in image `(c)`. This approach relies on an arbitrary representation of objects and image comparison.

In contrast to analyzing individual web elements, another approach to detecting web page breakage involves analyzing the entire web page. One example of this approach is the method presented by Macklon *et al.*, which focuses on detecting visual bugs in HTML5 `<canvas>` games [149]. This approach uses an internal representation of objects called Canvas Objects Representation (COR) to identify and compare visual elements of the game. The key part of the methodology was decomposing visual content of the web page into individual images, similar to breaking down a screenshot into its constituent visual elements. Figure 2.3 illustrates the approach in detecting visual mismatches between two images, `a` and `b`, with image `c` showing detected mismatches. By treating differences in web page similarity as web page breakage, this approach could be leveraged to detect the breakage of visual elements in a web page. In Chapter 3, we elaborate on our approach to measuring the similarity of visual rendering of a web page. Our approach focuses on all visual elements, not just those related to canvas. This is useful for identifying subtle changes that may not be apparent such as slight shifts in layout, color changes, or alterations in text content. Also, a standard approach is essential for visual rendering of web pages in the wild, because unlike the custom representation of COR objects in a canvas game, web pages vary in layout and design.

Nisenoff *et al.* conducted a study to understand how users perceive web page breakages [165]. The study involved a qualitative analysis of extension user reviews, GitHub issue reports, and a user survey. The authors note the diversity of reported issues and knowledge of users, emphasizing the need for easy ways to report breakage issues. They also suggested that further research is needed to find patterns in web page breakage. In Chapter 4, we discuss our approach to detecting web page breakage and our heuristics to classify usability issues that result from it.

2.3 Minimizing browser distinctiveness

As we have seen in Section 2.1.1, browsers were initially designed to be distinguished from one another due to how they rendered web pages. This was later addressed by the W3C, which approved standards such as HTML5 and CSS3 to ensure that web pages render consistently across all browsers. However, this did not prevent browsers from continuing to reveal information about the user's browsing environment that still distinguishes them from one another. For example, through the Navigator interface, browsers reveal information about the device's hardware, operating system, geolocation, the name of browser, its version, vendor, and list of installed plugins among other things. The diversity of this information makes browsers uniquely identifiable on the web through techniques such as 'browser fingerprinting' [61, 67]. These techniques become increasingly effective as the collected information becomes more diverse. The more distinctive the collected information from the browser or its environment, the more effective the fingerprint.

Previous studies addressed the issue of browser distinctiveness through two main approaches, as we will see in the following sections. The first approach involved investigating the user's browsing environment to identify information leaks that contribute to browser fingerprinting. Studies using this approach focused on determining the sources of these leaks to develop countermeasures. These studies analyzed the diversity and variability of User-Agent strings, examined fingerprinting techniques under different conditions, and developed techniques for fingerprinting browser extensions based on style modifications. The second approach aimed to minimize the distinctiveness of browsers by enforcing restrictions on access to information from the user's browsing environment. Studies using this approach focused on limiting the information that browsers reveal. These studies proposed techniques such as debloating browsers by removing non-essential features, using anti-fingerprinting browsers that disguise system and browser properties, and using information flow control mechanisms to restrict nonessential information flow. Proposed techniques also include using Shadow DOMs to isolate styles applied by extensions and implementing Content Protection Policies (CPPs) to control access to sensitive data by third-party scripts.

2.3.1 Analyzing leaks from the user's browsing environment

Kline *et al.* analyzed a comScore data set of over one billion User-Agent (UA) strings sent over two years [125]. Their analysis revealed that UAs provide a detailed view of client systems indicating the diversity of system configurations and applications. The study showed the variability of UAs and highlighted that popular UAs comprise 26% of all traffic, the rest, majority of which are unique constituting a high degree of diversity. They pointed out that this diversity could be a rich source of browser fingerprinting and could be used to track individual users. In Chapter 4, we explore the impact of the UAs and

other device information on the web and the implications of restricting that information to generic text.

Juarez *et al.* examined the effectiveness of fingerprinting techniques under various conditions such as browsing habits and Tor browser versions [114]. The key takeaway from this approach is the importance of restricting the amount of information that the browser reveals. Similar to this approach, dynamically altering the information from the user's browsing environment can make browser fingerprinting more challenging. This could involve regularly changing browser characteristics or randomizing certain information. However, evaluating the impact of this approach on user experience could be a much harder challenge. It requires testing the browser's functionality across various websites while dynamically altering the information it reveals and assessing any web page breakages that may arise. Further research is needed to determine the feasibility of this approach.

Laperdrix *et al.* developed a technique for fingerprinting browser extensions based on the style modifications that extensions make on web pages [135]. They analyzed CSS selectors used by extensions, and managed to detect the presence of those extensions on web pages. This information can be used to improve browser fingerprinting. As a countermeasure, the authors suggested using Shadow DOMs in browsers to isolate styles applied by extensions from the main page's DOM. This means that styles injected by extensions would not be visible by the main web page hence masking the presence of browser extensions. This approach restricts information available in the user's browsing environment and can be used as a defense against browser fingerprinting.

2.3.2 Restricting access to the user's browsing environment

Qian *et al.* introduced Slimium, a framework designed to identify and eliminate non-essential features in the Chromium browser [182]. The authors employed a feauture-code mapping technique to identify features that can be safely removed without impacting the browser's functionality. The study demonstrates that debloating a browser not only reduces its complexity but also lowers the number of vulnerabilities which reduces information leaks thereby enhancing user privacy. While the study provides a comprehensive approach in debloating unnecessary features from Chromium, the framework assumes that the features being debloated remain consistent over time. However, web content and browser features are dynamic and continually evolve. Changes in web standards, browser functionalities, or even website designs might render a debloated browser less effective or even dysfunctional over time. By modifying the browser's codebase, there is a risk of introducing new vulnerabilities. Furthermore, the study focuses on the Chromium browser, which, while widely used, is just one in many browsers available. The methodologies and the findings in this study might not be applicable to other browsers.

Baumann *et al.* presented an anti-fingerprinting browser, Disguised Chromium Browser (DCB), protecting against Flash and canvas fingerprinting without deactivating fea-

tures [12]. DCB used real-world data to disguise system and browser properties, maintaining usability while preventing detectability through unrealistic or constantly changing parameters. For example, DCB modified the canvas element in every session, making it unique and undetectable, ensuring the fingerprinter cannot detect modifications or re-identify users. This approach counters fingerprinting techniques using graphic rendering engines and Unicode glyphs.

Datta *et al.* evaluated the ability of Anti-Fingerprinting Privacy Enhancing Techniques (AFPET) in masking or spoofing browser attributes used in fingerprinting [51]. The authors evaluated various AFPETs including the Tor Browser Bundle, and found inconsistencies in the behaviors of some AFPETs. The Tor browser was identified as the most effective in masking attributes while other AFPETs either left several attributes unmasked or behaved inconsistently. The study's findings highlight the importance of effectively masking browser attributes as a defense technique against fingerprinting. This is a good addition to the landscape of information restriction as a means to enchance user privacy. Future research should focus on improving browser attribute masking techniques to limit the leakage of nonessential information.

Starov *et al.*analyzed 58,034 Google Chrome extensions to determine how extensions become fingerprintable through unnecessary page modifications termed, extension bloat, and found that 5.7% of those extensions are unnecessarily identifiable due to bloat [210]. To reduce fingerprintability, the authors proposed an in-browser mechanism providing coarse-grained access control for extensions to protect against fingerprinting and malicious extensions exfiltrating user data.

Vastel *et al.* developed FP-Scanner, a tool to detect alterations in browser fingerprints caused by countermeasures such as script blockers and User-Agent spoofers, introducing inconsistencies that would not otherwise occur [233]. By analyzing browser attributes such as HTTP headers, user agent, WebGL, canvas, and the JavaScript, FP-Scanner identified altered browser fingerprints. This suggests that many privacy-enhancing tools may unintentionally make browsers more uniquely identifiable instead of less. The study underscores that restricting nonessential information is not straightforward, as efforts to mask or alter information can create new privacy risks. This highlights the need for restricting nonessential information without inadvertently exposing new information by creating patterns or inconsistencies that could be used for fingerprinting. This is why detecting web page breakage, as discussed earlier and later in Chapter 4, is important to prevent further increasing the risk of leaking information from the user's browsing environment rather than preventing it.

Torok *et al.* introduced Sandcastle, a browser fingerprinting defense technique focusing on minimizing interference between the fingerprinting process with legitimate web applications [220]. Their approach was to partition code dealing with identifiable information into sandboxes, ensuring that sensitive information is not transmitted over the network while still being usable on the client-side. This method involved a detailed structure

for the sandboxing process including specific rules for data handling, computation, and communication between the sandbox and the main application context. By confining high-entropy data to client-side operations and preventing it from being sent over the network, Sandcastle could reduce the information leaks, thus providing a solution to the challenge of maintaining web pages functional while also respecting user privacy. However, its shortcomings lie in the sandboxing techniques against code obfuscation. Furthermore, future assessments are needed to examine the behavior of different browsers across different websites.

Senol *et al.* investigated the effects of Google Chrome's UA reduction efforts on exposing potentially identifying browser features using an instrumented cralwer to analyze JavaScript calls, HTTP headers, and HTML elements related to UA client hints (UA-CHs) [200]. The authors quantified access to high-entropy browser features through UA-CH HTTP headers and JavaScript API, focusing on the access of third-party scripts to these features. The study showed that scripts from third-party domains retrieve high entropy UA-CHs, indicating widespread use by trackers and advertisers exfiltrating those hints to remote servers by tracker scripts. The study concluded that UA reduction efforts have been effective in minimizing the passive collection of identifying browser features, but that third-party tracking and advertising scripts continue to have unrestricted access on those features.

Xiao *et al.* identified vulnerabilities through hidden properties abusing (HPA) in Node.js applications where unexpected properties can lead to information leaks [254]. The study implies that restricting nonessential information in Node.js requires examination of how objects are shared and manipulated between client and server sides. To address these leaks, there could be an implementation of stringent checks on the properties of the objects being processed, ensuring that only essential properties are allowed to interact with the system's core functionalities. In Chapter 3, we develop a tool to examine such scenario where the relevance of information can be tested using web similarity. As we further explore in Chapter 4, if such information does not break the web page, it can be deemed nonessential.

Wang *et al.* introduced Content Protection Policy (CPP) focusing on fine-grained confidentiality and integrity protection for sensitive client-side user data in web applications [246]. The study targets the issue of sensitive data exposure by identifying a gap in existing browser security policies which restrict script execution but do not limit access to sensitive web content by third-party scripts. The authors developed DOMinator, a system integrated into the browser to ensure that all script interactions with the DOM are checked for permissions by applying the least privilege principle. Unlike conventional approaches, CPP operates at an object level, allowing precise control over individual data elements. This granularity is important in restricting nonessential data exposure. By allowing control over what data third-party scripts can access, CPP reduces the risks of unintentional information leaks. This makes this approach useful in enhancing user privacy

through nonessential information restriction, given the increasing reliance complexity of web applications and the widespread use of third-party scripts. However, it requires widespread adoption by browser vendors and the web community and thus its challenging feasibility.

In conclusion, the issue of browser distinctiveness has been addressed through various approaches, ranging from investigating the user's browsing environment to identify information leaks to enforcing restrictions on access to information from the user's browsing environment. While these methods have shown promise, they often fall short in providing comprehensive solutions, either by being overly ambitious or only offering limited solutions. A phased approach is necessary, with research conducted on a case-by-case basis to explore the relevance of various information in the user's browsing environment. In this thesis, we have developed a tool that provides a framework for analyzing that relevance. This tool serves as the backbone for conducting research on browser distinctiveness, eliminating the need to develop a new tool for each study. Using this tool, we have explored the impact of device information on the web, and Chapter 3 will delve into that.

2.4 The geolocation impact on privacy policies

Privacy policies have become increasingly lengthy, complex, and difficult to understand over time, with their effectiveness being subject to scrutiny [145, 6, 209, 152]. Early studies of the regulatory landscape on the web have cast doubt on the ability of privacy regulations to change the behavior of websites, beyond requiring them to display cookie consent notices [54, 207]. Furthermore, there is a question of what happens when users are located in countries that are not covered by these policies. Previous studies indicate that web behavior remains unscathed regardless of geolocation. For example, Sørensen et al. claimed that user profiling has become a standard product offered for sale to any advertiser and that third-party activities did not change as a result of privacy regulations [208]. Another method that has not been hindered by privacy regulations is cookie synchronization [171, 223, 224]. In this section, we investigate the impact of geolocation on privacy policies across countries, focusing on techniques to identify cookie consent notices and interaction with those notices. We classify previous studies into three categories: human-assisted, filter-list, and z-index based methodologies. Additionally, we examine the advantages and disadvantages of existing approaches for interacting with cookie consent notices.

2.4.1 Identifying cookie consent notices

When a user sees a cookie consent notice, they usually notice a popup up or a highlighted section of the web page with words like 'policy', 'cookie', 'privacy', etc. While that is easy for the user to understand, it is difficult to simulate the user's perception. That simulation

must recognize the notice on the web page as a cookie consent notice or privacy policy, and not mistake it for something else like a discount offer, an email subscription, or an age verification pop-up.

Some of the previous studies mentioned below used human-assisted approaches to reduce bias in their analyses. Others used filter lists, which are known to be used by consent management platforms (CMPs). This approach assumes that certain patterns of elements on the web page indicate the presence of a CMP, and therefore, the website presents cookie consent notices to the users. However, not all websites use CMPs for consent management. Another approach is to calculate the superpositioning of elements on the web page using `z-index` and look for keywords in elements at each position. This approach is limited by two assumptions:

1. The web page is designed in conventional ways, such that consent text can only be expected in certain elements, which is not always the case on the web.

2. The consent text is predictable and can be looked up from a set of words.

Unfortunately, the second assumption continues to be a challenge for any approach to automatically detect cookie consent notices. Furthermore, the existing studies have not addressed the scenario where a website may selectively display cookie consent notices based on the user's geolocation. For instance, a website may only show the notice to users located in countries with privacy regulations, while users in other countries may not see the notice at all. This limitation highlights the need for further research to develop methods for detecting cookie consent notices across different countries.

Human-assisted methodologies Gray *et al.*analyzed different types of consent notices, focusing on how they are designed and their implications for user privacy consent using an interaction criticism approach integrating perspectives from Human-Computer Interaction (HCI) studies [93]. The authors analyzed the intent of the designer, the designed interface, and the social impact, providing a comprehensive view of how consent mechanisms work and their potential pitfalls. The study identified various dark patterns in consent notices such as tracking walls that manipulate user consent. These patterns often obscure or complicate the process of refusing consent, pushing users towards agreeing to data collection and processing unknowingly or unwillingly. The study highlighted legal and ethical issues surrounding these practices, especially in compliance of regulations such as GDPR. In Chapter 5, we explore the landscape of cookie consent notices across different countries, which may highlight differences in intent and purpose across regions.

Santos *et al.* conducted a study that examined both the design and textual content of cookie consent notices and evaluated their compliance with privacy regulations [197]. The authors manually annotated cookie banners on English-speaking websites visited by users residing in the EU. To detect cookie banners, they followed a three-step approach: segmentation, scoring, and tree traversal. First, they segmented webpages into small

segments and built a segment tree based on the HTML tag and text in the segments. Second, they assigned a score to each segment based on its inner text using a vocabulary set that they created by analyzing cookie banner content. They ranked tree leaf segments according to their scores. Third, they used the highest-scoring segments to traverse their segment tree, performing both bottom-up and top-down tree traversals. They captured HTML elements that contained cookie banners. To reduce false positives like websites with no banners, they used the segment scores to decide whether a cookie banner existed based on a threshold they set. Finally, they manually filtered out any remaining false positives. The study identifies common issues in cookie banners, such as vague language, technical jargon, and misleading statements, which can obscure the true purpose of data collection, thereby impeding the ability of users to make informed decisions. While this is a good approach, it may be difficult to scale in terms of the size of analyzed websites and beyond the English language.

Filter lists methodologies Nouwens *et al.* developed Consent-O-Matic, a browser extension that answers cookie consent notices based on user preferences, bypassing non-compliant interfaces [167]. This achieved this by designing an interoperability of CMPs in the Consent-O-Matic extension. The study highlights a gap in web privacy where the rights of the user are often overlooked due to ineffective enforcement of GDPR. By automating consent based on user-set preferences, Consent-O-Matic challenges the practices of CMPs of framing default positive choices for the users and serving consent mechanisms as a formality.

Another study by Kampanos *et al.* investigated the compliance of cookie consent notices to privacy regulations, and their implications on user privacy in Greece and the UK [116]. This is based on the IDCAC extension [124]. The study found discrepancy between the number of websites using third-party cookies and those displaying cookie consent notices, indicating non-compliance with privacy regulations such as GDPR. This highlights the widespread issue of overlooking user choice leading to information leaks and making users vulnerable to tracking. Furthermore, the study provided comparative analysis into how privacy practices vary between countries under similar legal frameworks. For example, cookie consent management in Greece showed higher compliance and transparency compared to the UK where there is more tracking.

Bollinger *et al.* introduced CookieBlock, a browser extension that uses machine learning to enforce GDPR cookie consent notices [19]. CookieBlock relies on CMPs to label the purpose of collected cookie consent notices. This ground truth data is then used to train classifiers to categorize cookies based on their purpose. The idea is that users can set preferences for which types of cookies to allow or decline, and CookieBlock will enforce automatic filtering of cookie consent notices based on those user-set preferences. The study highlights a gap in GDPR compliance, with many websites failing to provide legally valid consent options for cookies.

Z-index methodologies Khandelwal *et al.* proposed CookieEnforcer, a system for automated analysis and enforcement of cookie consent notices on websites [120]. The objective of the study was to addressed the issue of cookie consent notices designed to manipulate users into consenting to privacy-compromising settings. The authors used a z-index approach to detect the presence of a notice, and then used a BERT [55] text classifer to determine that the type of the notice is a cookie consent. The fact that this study only examined 250 and required manual annotation to validate the presence of cookie consent banners limits its scalability and feasibility.

Rasaii *et al.* conducted a measurement study to analyze the cookie landscape from different geographic locations using BannerClick, a tool they developed to automatically interact with cookie consent notices [186]. The study revealed how cookie practices vary depending on the user's geographic location with significant differences between EU and non-EU regions such as the increase in third-party cookies and tracking. Despite the reported accuracy of BannerClick in interacting with the consent notices, the tool is limited to 12 languages and the explicit consent notice text, making it ineffective for use in the wild where there is a large number of languages and nuances in consent notices.

Methodologies based on z-index present limitations in their reliance on HTML rendering patterns when considering their application in real-world scenarios. One of those limitations is their ability to capture the full diversity of text in cookie consent notices across the web. Many websites employ techniques to generate dynamic content and complex layouts which can result in cookie notices that do not conform to standard rendering patterns leading to cases where methodologies that rely on HTML rendering patterns failing to detect the cookie consent notice.

For example, Listing 2.1 shows what we observed on websites that use React to manipulate the DOM, creating elements on the fly upon user interactions and other triggers. In this example, the cookie notice `<div className="cookie-consent-banner">` is rendered based on the `showNotice` state variable. If this notice is not present at the initial page load, the HTML rendering patterns will miss it, leading to a false negative in detecting the notice. In Chapter 5, we discuss such cases where the same web page displays a cookie consent notice only to users from regulated areas.

Listing 2.1: Cookie consent notice generated on the fly by React

```
function CookieNotice() {
  const [showNotice, setShowNotice] = useState(false);

  useEffect(() => {
    // notice is rendered based on a state variable
    setShowNotice(checkCookieConsent());
  }, []);

  if (!showNotice) return null;
```

```
  return (
    <div className="cookie-consent-banner">
      We use cookies for better user experience. <button onClick={handleConsent
          }>Accept</button>
    </div>
  );
}
```

Websites also use HTML or CSS to alter the appearance of consent notices. In this case, as shown in Listing 2.2, the notice is hidden off-screen by the CSS property `transform: translateX(-100%)` and can only be displayed by a condition from the JavaScript function `acceptCookies()`. This could be the case where a website selectively displays a consent notice only when the user is located in a regulated country.

Listing 2.2: Cookie consent notice altered by JavaScript

```
<div id="unique-consent-model" style="transform:␣translateX(-100%);">
  <div class="custom-consent-banner">
    We use cookies for better user experience. <a href="javascript:void(0);"
        onclick="acceptCookies()">Agree</a>
  </div>
</div>
```

Another limitation in detecting cookie consent notices pertains to the use of machine learning models. While the studies demonstrate high accuracy in controlled test scenarios, these models are difficult to implement in real-world scenarios with the evolving nature of web technologies and privacy policies. Websites update their interfaces and the wording of cookie notices, which render previously trained models less effective. This necessitates continuous training and updating of the models, a process that is resource-instensive and does not always keep up with the rate of change of the web as we have seen in Chapter 5 with CookieBlock [19]. In Chapter 5, we introduce a novel approach for automatic visual detection of consent notices, thereby addressing the limitations of relying solely on manual observation, CMPs, or HTML/CSS components.

2.4.2 User interaction with cookie consent banners

Previous studies have highlighted violations in cookie consent mechanisms, which any study on the impact of cookie consent notices should carefully consider to avoid a biased evaluation of tracking based on different user choices.

Utz *et al.* conducted experiments on a German website with over 80,000 unique users to to examine the design and effectiveness of cookie consent notices in relation to GDPR and user understanding of privacy choices [227]. When users were given a binary choice (accept or decline), more opted to accept tracking. In contrast, when

required to approve cookie use for each category individually, users were more likely to decline. This indicates that simpler choices might lead to increased acceptance of nonessential information collection. However, this may also be because simpler choices include mandatory options such as essential cookies for the website to function. The study also discusses user expectations and misconceptions about consent notices. Many users mistakenly believe that not interacting with a consent notice will prevent websites from collecting data. This misunderstanding underscores the need for clearer communication in consent mechanisms, ensuring users are truly informed about what constitutes the collected information and how it is handled.

Matte *et al.* studied the compliance of cookie consent notices with legal regulations such as GDPR and the ePrivacy Directive [153]. They used the Interactive Advertising Bureau (IAB), EU's Transparency and Consent Framework (TCF), to detect legal violations in the implementation of these notices on European websites. The authors identified four types of potential legal violations: consent stored before choice, no way to opt-out, pre-selected choices, and non-respect of choice. These violations were observed in various proportions across the websites, highlighting widespread issues in the practice of obtaining user consent for cookies and tracking. While the methodology in this study is limited to consent notices compliant to TCF, it can be used as a benchmark for evaluating the effectiveness of consent management platforms (CMP).

O'Connor *et al.* conducted a user study to understand how websites comply with the California Consumer Privacy Act (CCPA) [168]. They found that most websites, instead of providing clear cookie consent notices with an option to decline, implement a "Do Not Sell" link. However, this approach often presents users with lengthy forms or dark patterns, making it difficult to decline consent for the use of cookies. The authors conducted manual observations to examine the impact of this behavior on users. They found that users often fail to decline consent and become less aware of their right to do so.

Sanchez-Rola *et al.* found that the GDPR has influenced non-EU websites to adopt similar practices to EU websites [196]. However, the study revealed persistent tracking, often without clear user consent, and noted the prevalence of cookies capable of identifying users across the majority of websites. Furthermore, the authors uncovered numerous instances of misleading information presented to users, making it challenging to refuse cookie consent.

Santos *et al.* highlighted the use of vague language and framing techniques in cookie consent notices [198]. They used an interdisciplinary approach to evaluate cookie consent notices against the ePrivacy Directive and the GDPR with a focus on the textual content of the notices. Their methodology involved a three-step process: segmenting web pages based on HTML tags and text, assigning scores to segments based on a vocabulary set, and traversing the segment tree to capture HTML elements containing cookie consent notices. Their analysis revealed that 89% of the consent notices violated at least one legal requirement, while 61% violated the purpose specificity requirement by using vague

language such as “user experience enhancement". The authors suggest standardizing the purposes of cookie consent notices and improving the clarity of data processing descriptions.

Dimova *et al.* conducted a longitudinal analysis of Facebook's cookie-based tracking from 2015 and 2022 [57]. The authors examined how practices of setting cookies for both users and non-users have evolved in response to privacy regulations such as GDPR. They found that Facebook initially set cookies automatically when non-users visited the website, leading to tracking across websites with Facebook resources. Over time, stricter regulations prompted a reduction in such practices. For registered users, Facebook introduced more granular controls over cookie settings, including choices for optional cookies related to advertising and third-party cookies. However, the study noted the use of dark patterns in consent mechanisms, which can influence users towards more invasive privacy choices.

Iordanou *et al.* developed a browser extension to analyze cross-border tracking flows and geolocate trackers in relation to the geolocation of users [113]. The authors collected data from the interactions of 350 real users with cookie consent notices and combined it with ISP datasets. This allowed them to identify and geolocate web trackers from captured tracking flows. They found that most tracking flows within the EU remain confined within the GDPR jurisdiction, suggesting that EU data protection laws effectively regulate these flows. However, the study does not discuss the prevalence of cookie consent notices and the state of tracking in websites across countries, regardless of the geolocation of the web trackers.

Wesselkamp *et al.* developed a browser extension, ERNIE, to detect cookie-based tracking techniques [251]. The authors focused on moments before and after user interaction with cookie consent notices. Their analysis included technical and legal assessments of tracking techniques in the context of GDPR. The study found tracking activities before any user interaction with consent notices and after rejection of the consent by users. We adopt the methodology of detecting ID cookies from this study into our own approach of measuring the state of cookies, as discussed in Chapter 5. These ID cookies are used by websites and third parties to identify users across sessions and websites. The methodology to detect ID cookies in this study involves comparing cookies between a main user session and a simulated separate user to determine if a cookie is user-specific.

Papadogiannakis *et al.* employed an automated method to detect first-party ID leaking, ID synchronization, and browser fingerprinting in their investigation of how websites track users even when they do not consent or choose to reject cookies [170]. The study revealed widespread first-party ID leaking and third-party ID synchronization, indicating that many websites do not fully comply with GDPR as they continue to track users without proper consent. In Chapter 5, we conduct a similar investigation to explore the state of tracking on the same websites across different countries and observe differences.

Munir *et al.* addressed the shift in web tracking from third-party to first-party cookies due to the increasing prevalence of browsers blocking third-party cookies [159]. They proposed CookieGraph, a machine learning-based method to detect and block first-party tracking cookies. The authors demonstrated that first-party cookies are being used to store and share identifiers with trackers, even when third-party cookies are blocked. The study highlighted the ineffectiveness of solely blocking third-party cookies in preventing tracking and the impracticality of blocking all first-party cookies, as it would result in web page breakage.

The variation in user tracking across countries with different levels of privacy regulations has been reported on previously [80]. However, the methodology used to measure the prevalence of cookie consent notice enforcement and user interaction with those notices has not been comprehensively addressed. Studies on this topic have limited geographic coverage and do not consider the interplay of tracking across locations for the same users. We considered the challenges in the studies above when designing our approach to user interaction with cookie consent notices, which we discuss in detail in Chapter 5.

Chapter 3

Determining information relevance in the user's browsing environment

3.1 Overview

The user's browsing environment reveals a lot of information about the device and the browser. This information raises privacy concerns, as its diversity allows for the identification of users on the web without their consent through fingerprinting techniques [132]. To enhance user privacy, it is essential to evaluate the necessity of sharing this information with websites. Such an evaluation can help minimize data exposure by only sharing information that is essential for website functionality. To achieve this, the browser must be able to assess the relevance of each piece of information that the website is trying to access.

In this chapter, we propose a novel approach to determine that relevance by simulating website visits from a normal user's browsing environment versus restricted environments. We design web similarity techniques to observe differences in website behavior when various restricted environments are used and the information in those environments that cause the changes. If a piece of information is removed from the used environment and there is no difference in website behavior, it indicates that the removed information is irrelevant to website functionality and therefore unnecessary. On the other hand, if differences are found, a change impact analysis can help further determine the type of change and its severity to website functionality.

In the following sections, we delve into the details of our approach and the resulting web similarity tool. First, we review various types of information in the user's browsing environment and formalize our objective. We then discuss the simulation of the user's browsing environment. Next, we show the crawl process in experimenting with the simulated user's browsing environment. We then elaborate on the design of our web similarity, including the choice of web similarity dimensions to investigate, the use of similarity radar, the implementation details and how we determine the relevance of

information in the user's browsing environment. Finally, we discuss the threats to the validity of our tool and potential future improvements before concluding this chapter.

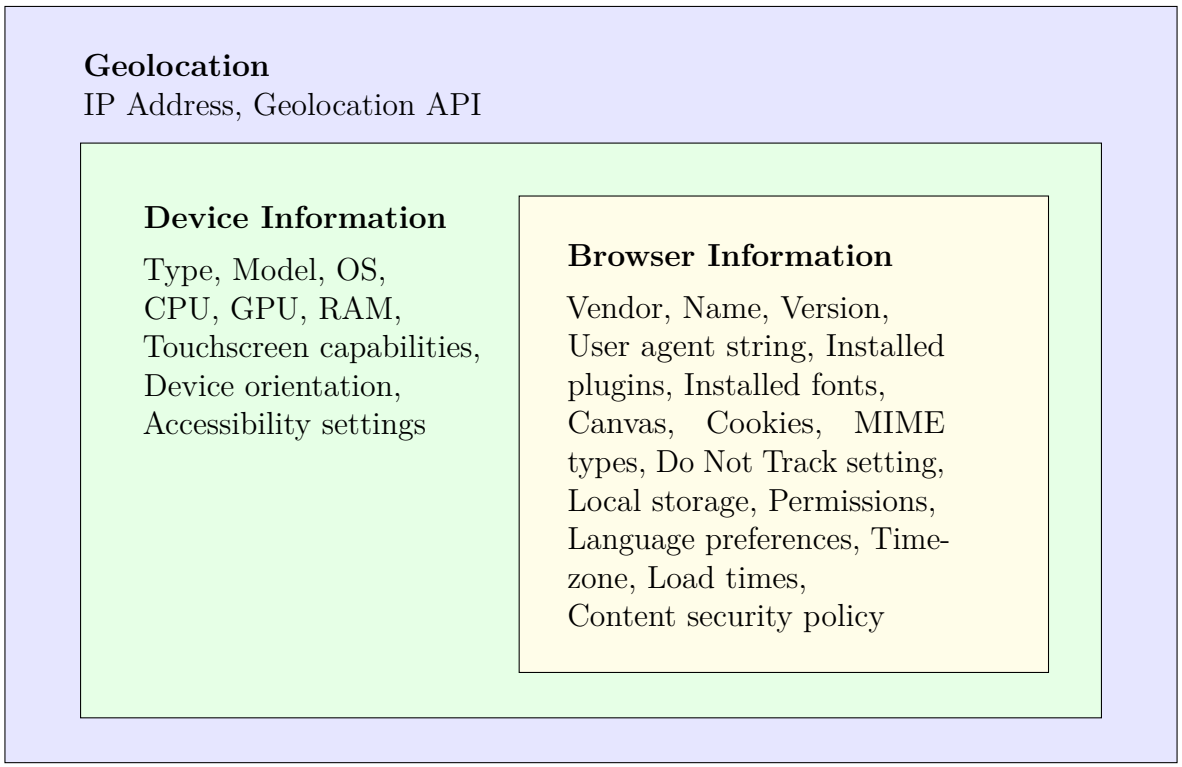


Figure 3.1: Categories of UBE information: Geolocation, Device, and Browser Information

3.2 What is UBE information?

The information in the user's browsing environment, which we refer to as "UBE information", can generally be categorized into geolocation, device, and browser data. This information encompasses various aspects of the user's hardware, software, and configurations. Websites access this information through HTTP headers or browser APIs via JavaScript. Additionally, server-side technologies can reveal UBE information by analyzing network and HTTP requests [158, 245, 213, 151]. In this chapter, we focus on methods of UBE access via JavaScript and HTTP headers. Previous works have explored UBE information extensively, and our experiments build upon those foundations [235, 237, 233, 89, 9, 8]. Figure 3.1 summarizes UBE information. Our objective is to determine the relevance of this information when websites access it, not to exhaustively list it. Geolocation information allows websites to determine the user's country and timezone, which in turn influences the experience and privacy compliance measures provided to the user [80]. Device information helps websites decide on the appropriate layout and functionality [162, 259], while browser information determines the specific features the website can use [179, 133].

A UBE construct $u = G \cup D \cup B$ represents sets of attributes G, D, B corresponding to geolocation, device, and browser information respectively. The set $G = \{g_1, g_2\}$ includes attributes that define the user's geographical location where g_1 and g_2 represent the IP address and the browser geolocation data obtained via the Geolocation API respectively. The set $D = \{d_1, d_2, ..., d_k\}$ includes attributes that characterize the user's device where d_i represents device information such as its type, model, OS, CPU, GPU, RAM specifications, touchscreen capabilities, device orientation, accessibility settings, etc. The set $B = \{b_1, b_2, ..., b_k\}$ includes browser attributes where b_i represents information such as the browser vendor, name, version, plugins, etc. as illustrated in figure 3.1.

To determine the relevance of information accessed by a website, we introduce a modified UBE construct $u' = G' \cup D' \cup B'$ which simulates restricted access by partially populating or completely omitting specific attributes. G', D', and B' are the modified sets of attributes for the geolocation, device, and browser information respectively such that $G' \subseteq G$, $D' \subseteq D$, and $B' \subseteq B$. The set u' allows simulating different levels of data exposure in a controlled manner. For example, a construct u' where $G'=\{g_i\}$, $D'=\emptyset$, and $B'=\{b_i\}$ can be a simulation of a UBE where only the IP address and the user agent string are exposed to the website.

We define a function $f : u \to r$ that maps the UBE construct u to a result r representing the behavior of the website when visited using UBE construct u . Similarly, we define $f' : u' \to r'$ when the same website is visited using a modified UBE construct u'. The difference in website behavior $\Delta r = |r - r'|$ between r and r' determines the relevance of the modified attributes. If $\Delta r \neq 0$, further analysis can indicate the type of change and its severity. We will explore this in Chapter 4. If $\Delta r = 0$, it indicates that the difference $\Delta u = u - u'$ between the original UBE construct u and the modified construct u' is irrelevant to the website's functionality. Attributes in $\Delta u = \{x_1, x_2, ..., x_k\}$ such that $\Delta u = \{x \in u \mid x \notin u'\}$ can therefore be restricted without negatively impacting the website's functionality.

We analyze the impact of each attribute x_i in Δu by considering whether the removal of each attribute affects the overall difference in website behavior Δr. Specifically, we define the binary impact function $\delta : \Delta u \to \{0, 1\}$ that indicates whether the removal of an attribute x_i has any impact on Δr such that:

$$\delta(x_i) = \begin{cases} 1 & \text{if } f(u) \neq f(u \setminus \{x_i\}) \\ 0 & \text{if } f(u) = f(u \setminus \{x_i\}) \end{cases} \tag{3.1}$$

where $u \setminus \{x_i\}$ represents the UBE construct u with the attribute x_i removed. If $\delta(x_i) = 0$ for a specific attribute x_i, this attribute does not affect the functionality and can be considered irrelevant. Thus, the set of irrelevant attributes $I \subseteq \Delta u$ is defined as $I = \{x_i \in \Delta u \mid \delta(x_i) = 0\}$. Conversely, the set of relevant attributes $R \subseteq \Delta u$ is defined as $R = \{x_i \in \Delta u \mid \delta(x_i) = 1\}$. Our objective is to determine a minimal set R and a maximal

set I while maintaining the website's functionality. This would ultimately lead to a new UBE construct $u'' \subseteq u$ that balances functionality and web privacy.

3.3 Simulating access to UBE

To determine the UBE construct $u'' \subseteq u$, we need to test website visits using simulated UBE constructs to maximize the set I and minimize the set R. So UBE simulation is key to this process. The ideal way to simulate a UBE construct u' would be to physically visit different locations, install new computers, and access websites with a variety of browsers and configurations. This approach is impractical, however, due to resource and time constraints. To overcome these limitations, we alternatively turn to UBE simulation techniques. Methods such as browser instrumentation, browser extensions, script injection via developer tools, VPNs, and proxy servers provide controlled environments where specific UBE attributes can be modified or restricted. Network interception was also once valuable for this purpose, however, the increasing adoption of the QUIC protocol has rendered its future relevance questionable [130, 195]. Hence, the following sections explore the methods we mentioned previously of simulating UBE, except for network interception.

3.3.1 Simulating geolocation

The two most common methods to geolocate users are IP and browser-based geolocation [118, 211]. IP geolocation determines the user's approximate location, typically at the city level, by looking up the IP address in a geolocation database. Browser geolocation uses the Geolocation API to request more precise location data directly from the user's device. This method prompts the user for explicit permission before accessing the device's location. Although IP geolocation can be inaccurate when the user uses IP masking techniques such as VPN or a proxy server [85], it is more widely implemented due to its simplicity and the fact that it does not require user interaction [122, 176]. However, both methods have their advantages and specific use cases. Of the existing UBE simulation techniques, only browser instrumentation can bypass the permission-granting process in simulating browser geolocation. Listing 3.1 illustrates an example of that override in Puppeteer [180]. The override not only can give websites access to the device's physical location but can also help simulate various geolocation information like the custom coordinates, as figure 3.2 shows. Apart from Puppeteer, other browser instrumentation tools like Selenium and Playwright can bypass the permission-granting process. This approach can also work in other UBE simulation techniques but will require user interaction to grant the permission.

In any scenario, G' always contains at least g_1 representing the IP address, since the web server receives it by default. Therefore, we simulate G' in a UBE construct u' with either $\{g_1\}$ if only the IP address is available in u' or $\{g_1, g_2\}$ if both the IP address and browser geolocation are accessible in u'. $G' \neq \emptyset$ because the empty set is not possible.

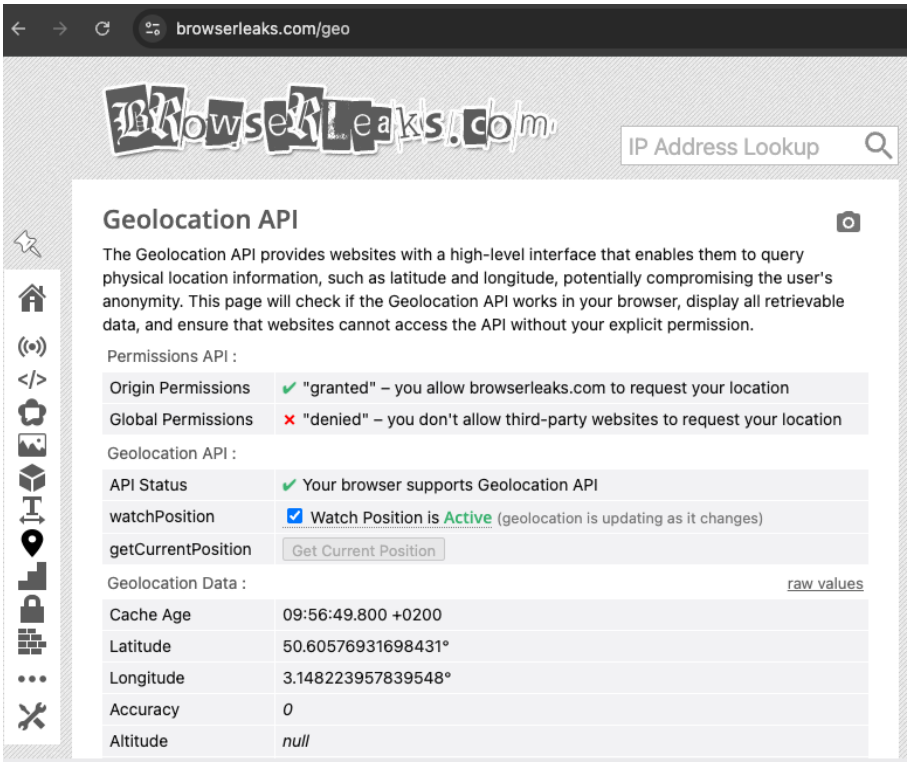


Figure 3.2: Browser geolocation exposing the user's coordinates due to bypassed permission-granting process in Puppeteer.

Apart from using a regular IP address (i.e. the normal, unmasked IP address of a user) to simulate $G' = \{g_1\}$, the two most common methods for simulating g_1 are VPN and proxy servers [75, 148]. VPN functions at the operating system level to ensure that all network traffic is routed to a designated endpoint. Proxy servers function at the application level, meaning that only G' (specifically the IP address) and B'-related browser attributes in UBE construct u' would be affected. This limitation can lead to inconsistencies across UBE constructs and network requests, hence the VPN is more reliable for simulating a consistent UBE. For designated areas, our UBE simulation uses VPN when $G' = \{g_1\}$ and both VPN and browser geolocation using browser instrumentation when $G' = \{g_1, g_2\}$.

Listing 3.1: JS Navigator.geolocation

```
import puppeteer from 'puppeteer';

(async () => {

  const browser = await puppeteer.launch({ devtools: true });

  const page = await browser.newPage();

  // Grants permission for changing geolocation
  const context = browser.defaultBrowserContext();
```

```
  await context.overridePermissions(
    'https://browserleaks.com/geo',
    ['geolocation']
  );

  // Changes to geolocation to custom coordinates
  await page.setGeolocation({
    latitude: 50.60576931698431,
    longitude: 3.148223957839548
  });

  await page.goto('https://browserleaks.com/geo/');

  ...
})();
```

3.3.2 Simulating device and browser information

Browser instrumentation

Browser instrumentation mimics real browsers and allows control over browser interactions, device emulation, and JavaScript execution. This makes it useful for handling websites that rely heavily on JavaScript, which is a big part of the web [79]. A unique feature of browser instrumentation is its ability to simulate user interactions, such as mouse movements, clicks, scrolling, keyboard input, and screenshot capturing. Another strength of browser instrumentation is its ability to bypass permission-based APIs like the Geolocation API demonstrated earlier. This capability extends to other browser APIs, enabling simulation of various UBE constructs.

Browser instrumentation tools such as Puppeteer, Playwright, and Selenium offer ways to simulate various devices [181, 173, 38]. This allows us to emulate attributes $\{d_1, d_2, ..., d_k\}$ of device D' in UBE construct u' without having to manually spoof every attribute. Browser instrumentation tools also allow configuring browser profiles in a way we can simulate attributes $\{b_1, b_2, ..., b_k\}$ of browser B' in UBE construct u'. These tools are extensive in emulating device and browser environments that, when coupled with our previous approach for simulating device geolocation, they give a flexible way to build lots of UBE constructs u' that can help us achieve our objective of determining a new UBE construct $u'' \subseteq u$ that balances functionality and web privacy.

While browser instrumentation tools are effective in simulating device and browser information, certain attributes are difficult to spoof convincingly. The presence of the `WEB_DRIVER` attribute, for example, or `navigator.deviceMemory` inconsistencies lead to these tools being detected by some websites as bots [234]. `WEB_DRIVER` is a flag

that indicates whether the browser is being controlled by an automation tool, while `navigator.DeviceMemory` is an attribute that indicates the approximate amount of device memory in gigabytes. In simulating the UBE construct u', we can overwrite these attributes, but this is not always foolproof. Some websites still find ways to identify browser automation [20]. Despite these limitations, however, browser instrumentation remains an effective method to simulate UBE constructs. Listing 3.2 demonstrates how to emulate an iPhone 15 in Puppeteer, and how to bypass basic detection by flagging off the `navigator.webdriver` attribute.

Listing 3.2: Puppeteer device emulation

```
import puppeteer from 'puppeteer';
import {KnownDevices} from 'puppeteer';

const iPhone = KnownDevices['iPhone␣15'];

(async () => {

  const browser = await puppeteer.launch({
    headless: false
  });

  const page = await browser.newPage();

  await page.evaluateOnNewDocument(() => {
    Object.defineProperty(navigator, 'webdriver', {
      get: () => false,
    });
  });

  await page.emulate(iPhone);

  await page.goto('https://www.responsivedesignchecker.com/');

  ...
})();
```

Browser extensions and developer tools

Browser extensions offer UBE simulation through JavaScript and browser APIs. They can modify HTTP headers, manipulate cookies, and inject scripts into web pages during website navigation. Although browser extensions can emulate device attributes such as screen size, and touch capability detection, they do not provide complete device

emulation. Unlike browser instrumentation tools, browser extensions cannot emulate hardware characteristics. They are limited to what is exposed by browser APIs and cannot simulate a complete UBE $u' = G' \cup D' \cup B'$. Browser developer tools on the other hand provide greater control over device emulation compared to browser extensions. However, like browser extensions, they are limited in their ability to emulate hardware characteristics and are confined to manual interactions. This limits their scalability for automated crawls. Both browser extensions and developer tools face significant limitations in simulating various UBE constructs for crawls at scale.

HTTP client libraries

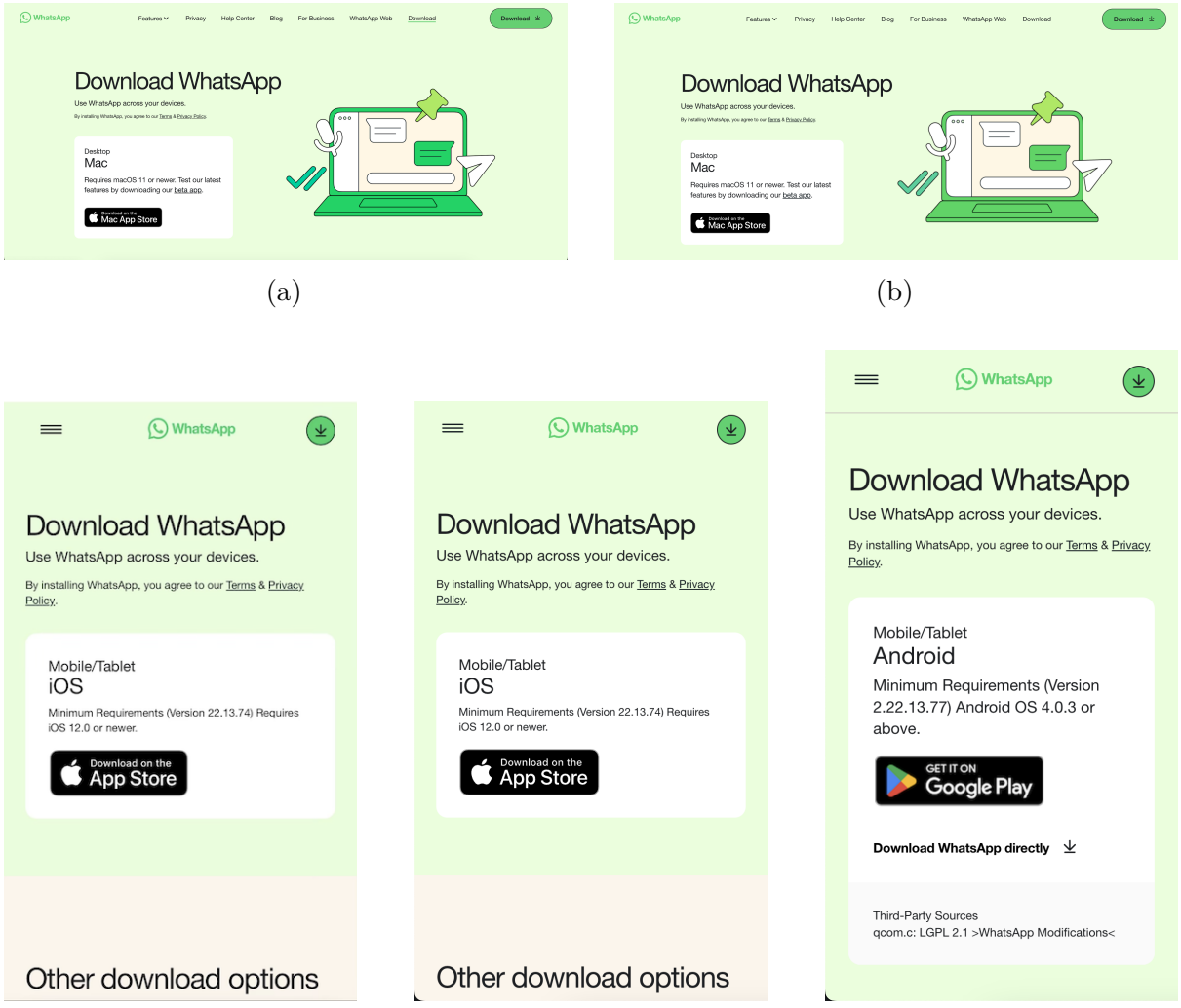
HTTP client libraries help configure and make HTTP requests to web servers. Tools like `cURL`, `wget`, or libraries in programming languages offer a scalable way to modify HTTP headers while crawling websites [247, 252, 188]. While this method can allow device emulation through user agent spoofing, it cannot simulate geolocation nor can it handle navigation of a web page including the execution of JavaScript. Listing 3.3 illustrates an example of the Python's `requests` library making an HTTP request that emulates an iPhone running iOS 15. While this may lead to receiving a response from a web server, the web page is not rendered, hence this method is very limited in simulating a complete $u' = G' \cup D' \cup B'$.

Listing 3.3: HTTP clients

```
import requests

headers = {
  'User-Agent': 'Mozilla/5.0␣(iPhone;␣CPU␣iPhone␣OS␣15_0␣like␣Mac␣OS␣X)',
  'Accept-Language': 'en-US,en;q=0.9'
}

response = requests.get('https://example.com', headers=headers)

# Further processing of the response
print(response.text)
```

In conclusion, for any experiment requiring control of UBE including rendering web pages, executing JavaScript, and simulating a UBE construct $u' = G' \cup D' \cup B'$, browser instrumentation is more effective than other simulation techniques (cf. Figure 3.3). Beyond simulating UBE, scalability is essential as we need to test as many UBE constructs as there are attributes in our UBE. This is necessary in order to achieve our goal of determining a UBE construct $u'' \subseteq u$ that balances functionality and privacy. Browser instrumentation also stands out compared to techniques like developer tools or browser extensions because it can be used at scale. We not only need to programmatically modify UBE constructs,

but we must also do so as we crawl from one web page to another. For this reason we opt for browser instrumentation, specifically Playwright, which is similar to Puppeteer but supports more browsers and can be used in multiple programming languages. In the next section, we discuss how we orchestrate crawls with the UBE constructs we have built. We elaborate on our crawl process in a way it can also be implemented using Puppeteer, Selenium, and any other browser instrumentation tools.



(a) (b) (c) (d) (e)

Figure 3.3: Comparing real devices with Puppeteer's UBE simulation while visiting `https://www.whatsapp.com/download/`: (a) real MacBook Pro, (b) simulated MacBook Pro, (c) real iPhone 13, (d) simulated iPhone 13, (e) simulated Galaxy S8.

3.4 Designing the crawler

Let $W = \{w_1, w_2, \ldots, w_n\}$ be the set of web pages to be crawled where each $w_i \in W$ represents a distinct URL. For each web page $w_i \in W$, there is a UBE construct u used in crawling the web page. Let $U = \{u_1, u_2, \ldots, u_m\}$ be the set of all UBE constructs,

where $u_j \in U$ is a distinct UBE construct. $u_j = G_j \cup D_j \cup B_j$ where G_j, D_j, B_j represent specific sets of geolocation, device, and browser attributes respectively. Our crawl process is described by a function $C : W \times U \to R$ where W is the set of web pages, U is the set of UBE constructs, and R is the set of data collected from the web pages during the crawl. For each web page $w_i \in W$ and each UBE construct $u_j \in U$, the crawler performs a request-response cycle and collects the data $r_{ij} = C(w_i, u_j)$ where $r_{ij} \in R$ represents the response obtained from the web page w_i under UBE construct u_j. Each response r_{ij} contains an HTML document, JavaScript and CSS files, a captured screenshot of the web page, and data about cookies and HTTP requests.

Algorithm 1 Crawl web pages with different UBE constructs

```
1: function CrawlWebPages(W, U')
2:     R ← empty list                          ▷ Declare results list for responses
3:     for each w_i in W do                    ▷ Loop over each web page
4:         for each u'_j in U' do              ▷ Loop over each UBE construct
5:             r_ij ← SendRequest(w_i, u'_j)   ▷ Send HTTP request
6:             Add(r_ij, R)                    ▷ Record the response in the results list
7:     return R                                ▷ Return the list of responses
```

3.4.1 Crawl orchestration

Algorithm 1 outlines the process for crawling web pages W using UBE constructs U. We express the crawl responses $R = \{r_{ij} = C(w_i, u_j) \mid w_i \in W, u_j \in U\}$ in Equation 3.2 where the rows correspond to the web pages W and the columns correspond to UBE constructs U. Let M be the matrix of responses, where each element M_{ij} represents the response r_{ij} collected from web page w_i under UBE construct u_j.

$$M = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{bmatrix} \tag{3.2}$$

As the crawl is performed over time, we also consider the time factor in the process. The function $C_t : W \times U \to R$ represents the crawl performed at a time t where $C_t(w_i, u_j) = r_{ij}(t)$ is the response collected from web page w_i under UBE construct u_j. The crawl responses become $R(t) = \{r_{ij}(t) = C_t(w_i, u_j) \mid w_i \in W, u_j \in U, t \in T\}$, where each response $r_{ij}(t)$ represents the crawl result for web page w_i under UBE construct u_j at time t. T is the set of times at which the crawls are performed. The matrix becomes $M(t) = \{r_{ij}(t) = C_t(w_i, u_j) \mid w_i \in W, u_j \in U, t \in T\}$, where the rows correspond to the web pages W, the columns correspond to the UBE constructs U, and the elements $M_{ij}(t) = r_{ij}(t)$ represent the responses over time.

Algorithm 2 Determine URL failure based on $M(t)$

```
function CheckURLFailure(w_i, U, M(t))
    failed ← False                                  ▷ Initialize failure flag
    for each u_j in U do                            ▷ Loop over each UBE construct
        r_ij(t) ← M(t)[w_i][u_j]                    ▷ Get response for URL and u_j
        if r_ij(t) < 200 or r_ij(t) ≥ 400 then      ▷ Check if the response code is invalid
            failed ← True                           ▷ Mark the URL as failed
            break                                   ▷ Exit the loop if failure is detected
    return failed                                   ▷ Return whether the URL has failed
```

$M(t)$ and $R(t)$ help us validate our crawl by tracking the status of individual crawls for different web pages and UBE constructs over time. The matrix $M(t)$ provides granular control by mapping the status of each crawl at time t, enabling us to pinpoint which specific crawls may have failed. This helps in identifying incomplete or erroneous crawls that should be discarded (cf. Listing 2), preventing invalid data from being stored in our final results. The use of $M(t)$ ensures that we maintain data integrity by detecting errors in the crawl responses, such as non-OK HTTP status codes, timeouts, or other network failures. As the crawl is performed over time, $M(t)$ allows us to differentiate between valid and invalid responses within the overall crawl dataset $R(t)$. For example, if a web page fails to load or returns an error status code under a particular UBE construct at time t, this failure will be captured using $M(t)$, enabling us to discard this specific response from the valid set of crawl results (cf. Listing 3). After processing the responses using $M(t)$, we define the set of valid responses $V(t) = \{r_{ij}(t) \in R(t) \mid M_{ij}(t) \text{ indicates a valid response}\}$, which only includes the successful crawls from $R(t)$. This ensures that only valid crawl data is stored, maintaining the integrity of the dataset for further analysis.

Algorithm 3 Save or discard crawl data based on URL failures

```
function SaveOrDiscardCrawls(R(t), M(t), U, W)
    V(t) ← empty list                               ▷ Declare V(t) to store valid responses
    for each w_i in W do                            ▷ Loop over each URL
        failed ← CheckURLFailure(w_i, U, M(t))
        if failed then
            discard R(t)[w_i]                       ▷ Discard crawl data if URL failed
        else
            Add(R(t)[w_i], V(t))                    ▷ Add valid responses to V(t)
    return V(t)                                     ▷ Return list of valid responses in V(t)
```

3.4.2 Dealing with dynamicity of web pages

To address the dynamicity of web pages in our crawl data, we transform the set of valid crawl data $V(t)$ into the set of valid static crawl data $V'(t)$. This transformation involves removing dynamic content from HTML, JS, and CSS files stored during the crawl. To

achieve that, our crawl process is duplicated from start to finish, i.e. we run the crawl $C_t : W \times U \to R$ at a time t where $C_t(w_i, u_j) = r_{ij}(t)$ is the response collected from web page w_i under UBE construct u_j and we also run a simultaneous crawl $C_{t'} : W \times U \to R$ at a time t' where $C_{t'}(w_i, u_j) = r_{ij}(t')$ is the response collected from web page w_i under UBE construct u_j to address the dynamicity of web pages. Thus, for each web page w and UBE construct u, we have two sets of valid crawl data $v_{wu}(t)$ from $V(t)$ and $v_{wu}(t')$ from $V(t')$. We compare the HTML, JS, and CSS content between $v_{wu}(t)$ and $v_{wu}(t')$, and any difference in these contents is considered dynamic and discarded. This process results in $v'_{wu}(t)$, which contains only the static content that is identical in both $v_{wu}(t)$ and $v_{wu}(t')$. The aggregate of all $v'_{wu}(t)$ forms $V'(t)$, the valid static crawl data.

Algorithm 4 IdentifyDiffNodes

```
1: function IdentifydiffNodes(AST1, AST2)
2:     DiffNodes ← empty list                          ▷ Declare list for diff nodes
3:     Queue1 ← new queue containing root of AST1
4:     Queue2 ← new queue containing root of AST2
5:     while Queue1 is not empty and Queue2 is not empty do
6:         n1 ← Queue1.dequeue()
7:         n2 ← Queue2.dequeue()
8:         if NodesDiffer(n1, n2) then
9:             DiffNodes.append(n1)                    ▷ Add diff node to list
10:        Queue1.enqueue(n1.children)
11:        Queue2.enqueue(n2.children)
12:    return DiffNodes                                ▷ Return the list of diff nodes
```

Algorithm 5 RemoveDiffBlocks

```
1: function RemovediffBlocks(AST, DiffNodes)
2:     for all n in DiffNodes do                       ▷ Iterate over diff nodes
3:         b ← FindEnclosingBlock(n)                   ▷ Find enclosing block for node
4:         if b is not already in BlocksToRemove then  ▷ Check if block is already listed
5:             BlocksToRemove.append(b)                ▷ Add block to removal list
6:     for all b in BlocksToRemove do                  ▷ Iterate over blocks to remove
7:         RemoveNode(b)                               ▷ Remove the block from the AST
8:     return AST                                      ▷ Return the modified AST
```

Let $D = \{\text{HTML}, \text{JS}, \text{CSS}\}$ represent the dimensions of content we are considering, hence $c_d^{wu}(t)$ and $c_d^{wu}(t')$ are the content of dimension $d \in D$ extracted from $v_{wu}(t)$ and $v_{wu}(t')$. For each web page w and UBE construct u, we compute the intersection $c_d^{wu}(t) \cap c_d^{wu}(t')$ which contains the content that is identical in both crawls. The union over all dimensions d gives us $v'_{wu}(t)$, the static content for web page w and UBE construct u such that:

$$v'_{wu}(t) = \bigcup_{d \in D} (c_d^{wu}(t) \cap c_d^{wu}(t')) \tag{3.3}$$

Hence, the set of valid static crawl data $V'(t)$ is formed by aggregating $v'_{wu}(t)$ such that:

$$V'(t) = \{v'_{wu}(t) \mid w \in W,\ u \in U\} \tag{3.4}$$

We use a line-by-line comparison of $v_{wu}(t)$ against $v_{wu}(t')$ but the removal process ensures that $v'_{wu}(t)$ remains syntactically correct. To achieve that, we remove entire syntactic blocks where differences occur. For that, we need recourse to abstract syntax trees (AST). We parse the HTML, JS, and CSS files into their respective ASTs. For example, for the HTML parser, we do $\text{AST}^{wu}_{\text{HTML}}(t) = \text{Parse}\,(c^{wu}_{\text{HTML}}(t))$ and $\text{AST}^{wu}_{\text{HTML}}(t') = \text{Parse}\,(c^{wu}_{\text{HTML}}(t'))$. We use a similar approach for JS and CSS parsers. We then compare the ASTs to identify diff nodes which we remove and reconstruct the modified tree to obtain $v'_{wu}(t)$.

Algorithm 4 illustrates the process of identifying diff nodes in detail. Algorithm 5 details how, for each diff node, we remove the syntactic block containing that node from $v_{wu}(t)$. The function FindEnclosingBlock(n) traverses up the AST from node n to find the smallest syntactic block that can be safely removed (node b), and then RemoveNode(b) removes b and its descendants from the AST. In Chapter 4 we discuss the use of existing tools and previous works to implement this. After modifying the ASTs, we reconstruct the code files such that $c'^{wu}_{\text{HTML}}(t) = \text{Unparse}\,(\text{ModifiedAST}^{wu}_{\text{HTML}}(t))$ in the case of the HTML document. We use a similar approach in reconstructing the JS and CSS files. This approach ensures syntactic validity, so that in the next process of doing web similarity, parsers can process $V'(t)$ without encountering syntax errors. Below are examples of our syntactic block removal approach once a difference is found.

HTML

Original in $v_{wu}(t)$:

```
<div>
    <p>April 1, 2023.</p>
    <p>Welcome.</p>
</div>
```

Original in $v_{wu}(t')$:

```
<div>
    <p>April 2, 2023.</p>
    <p>Welcome.</p>
</div>
```

Resulting $v'_{wu}(t)$:

```
<div>
    <p>Welcome.</p>
</div>
```

Figure 3.4

JS

Original in $v_{wu}(t)$:

```
function getUserCount() {
    return 150;
}
var n = getUserCount();
```

Original in $v_{wu}(t')$:

```
function getUserCount() {
    return 155;
}
var n = getUserCount();
```

Resulting $v'_{wu}(t)$:

```
var n = getUserCount();
```

Figure 3.5

CSS

Original in $v_{wu}(t)$:

```
.header {
    background-image: url(
        'header_v1.png');
    color: #333;
}
```

Original in $v_{wu}(t')$:

```
.header {
    background-image: url(
        'header_v2.png');
    color: #333;
}
```

Resulting $v'_{wu}(t)$:

```
.header {
    color: #333;
}
```

Figure 3.6

3.5 Similarity Radar

As we advance toward analyzing our crawl data, it is essential to quantify how similar two instances of the same web page are, when crawled using varying UBE constructs. This is where the concept of Similarity Radar comes in. Similarity Radar allows us to compare two instances of the same web page w_i crawled at time t using two distinct UBE constructs u_j and u_k. The valid static crawl data $v'_{ij}(t)$ and $v'_{ik}(t)$ correspond to seven similarity dimensions such as the HTML structure, JavaScript, CSS, visual rendering, cookies, HTTP requests, and textual content. The radar's function is to provide a visual representation of the degree of similarity between these two responses across the seven dimensions. Each axis on the radar represents one dimension, and for each axis, a similarity score S_d is calculated based on how similar the two responses are in that specific dimension.

Therefore, we define S_d to represent the similarity score for dimension d and S_{score} to be the overall similarity score across all seven dimensions. For each dimension d, the similarity score between $v'_{ij}(t)$ and $v'_{ik}(t)$ is calculated as:

$$S_d(v'_{ij}(t), v'_{ik}(t)) = \frac{\text{matched elements in dimension } d}{\text{total elements in dimension } d} \times 100 \tag{3.5}$$

The score S_d ranges between 0 (completely different) and 100 (identical). The overall similarity score S_{score} is then calculated as the average of the similarity scores across all

dimensions where n is the total number of dimensions:

$$S_{\text{score}}(v'_{ij}(t), v'_{ik}(t)) = \frac{1}{n} \sum_{d=1}^{n} (S_d(v'_{ij}(t), v'_{ik}(t))) \tag{3.6}$$

Similarity Radar allows us to compute and visualize the similarity between two crawls. But beyond visualization, this score plays a role in identifying relevant and irrelevant attributes of the UBE construct. As we have seen in Section 3.2, $f(u)$ represents the function that maps a UBE construct u to the result of a web crawl. To analyze the relevance of an attribute $x_i \in u$, we use the similarity score S_{score} as a metric to assess how removing x_i from the UBE construct affects the web page behavior (cf. Equation 3.1). The similarity radar helps us compute $\delta(x_i)$ for each attribute $x_i \in u$, which ultimately leads to classifying attributes into $I = \{x_i \in \Delta u \mid \delta(x_i) = 0\}$ (irrelevant attributes) and $R = \{x_i \in \Delta u \mid \delta(x_i) = 1\}$ (relevant attributes). By iterating through all attributes $x_i \in u$, we compute which attributes do not affect web page behavior and can thus be removed from the UBE construct (the set I).

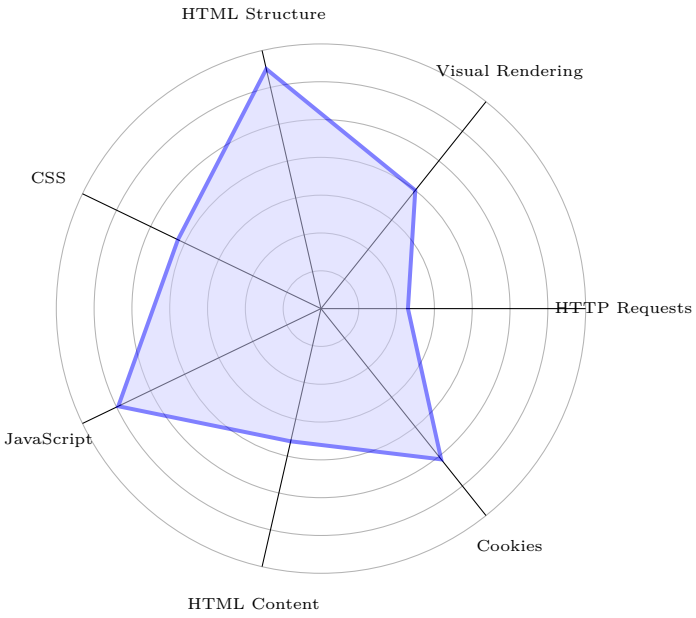


Figure 3.7: Similarity score S_{score} for two web pages $(v'_{ij}(t), v'_{ik}(t))$

3.5.1 Computing similarity scores for each dimension

We define methods to compute the similarity scores for each of the seven dimensions. Detailed implementation of our approach to computing the scores are in Chapter 4 for HTML structure, JS, CSS, visual rendering, and textual content. We provide more details concerning cookies and HTTP requests in Chapter 5.

HTML Structure

As mentioned in Chapter 2, to compare the HTML structure of web pages, we use the similarity-based flexible tree matching algorithm (SFTM) [23]. SFTM parses two web pages $v'_{ij}(t)$ and $v'_{ik}(t)$ into their respective DOM trees T_{ij} and T_{ik}. It then computes the similarity $S_0(n, m)$ between nodes $n \in T_{ij}$ and $m \in T_{ik}$. Finally, SFTM finds an optimal matching M between the nodes of the two DOM trees. We compute the similarity score S_{HTML} using SFTM's $c(M)$, which is the cumulative cost of matching nodes based on their similarities:

$$S_{\text{HTML}} = \left(1 - \frac{c(M)}{\max(|T_{ij}|, |T_{ik}|)}\right) \times 100 \tag{3.7}$$

JavaScript

JavaScript code in web pages is often distributed across multiple files and inline scripts. Comparing JavaScript code between two web page instances requires matching corresponding scripts and then computing their similarities. We begin by extracting all JavaScript files and inline scripts from both web page instances $v'_{ij}(t)$ and $v'_{ik}(t)$. Let $\mathcal{J}_{ij} = \{J_{ij}^{(1)}, J_{ij}^{(2)}, \ldots, J_{ij}^{(n)}\}$ be the set of JavaScript files from $v'_{ij}(t)$ and $\mathcal{J}_{ik} = \{J_{ik}^{(1)}, J_{ik}^{(2)}, \ldots, J_{ik}^{(m)}\}$ from $v'_{ik}(t)$. To match JavaScript files between the two sets, we first attempt to match files based on exact file names (URLs). For each file $J_{ij}^{(a)}$ in $\mathcal{J}_{ij}$, if there exists a file $J_{ik}^{(b)}$ in $\mathcal{J}_{ik}$ such that their URLs are identical, we consider them a matched pair. For JavaScript files that remain unmatched after exact name matching, we use Locality Sensitive Hashing (LSH) to find similar files based on their content [50]. LSH allows us to find files that are similar but have different names or inline scripts.

For each matched pair $(J_{ij}^{(a)}, J_{ik}^{(b)})$, we parse the code into ASTs and compare them using a tree-diff algorithm, GumTree [73]. The GumTree comparison returns the number of matched AST nodes and the number of edit operations required to transform one AST into the other. The similarity score for the pair is calculated as:

$$S_{\text{JS}}^{(a)} = \frac{\text{Number of matched AST nodes}}{\text{Total number of AST nodes}} \times 100 \tag{3.8}$$

For unmatched files, we treat them as having a similarity score of 0%. We compute the overall JavaScript similarity score S_{JS} where p is the number of matched file pairs:

$$S_{\text{JS}} = \frac{1}{p} \sum_{a=1}^{p} (S_{\text{JS}}^{(a)}) \tag{3.9}$$

CSS

The comparison of CSS files follows a similar process to that of JavaScript. We match CSS files based on exact file names (URLs). We use LSH to find similar CSS files with

different names or inline CSS. For each matched pair $(C_{ij}^{(a)}, C_{ik}^{(b)})$, we parse the code into ASTs and compare them:

$$S_{\text{CSS}}^{(a)} = \frac{\text{Number of matched AST nodes}}{\text{Total number of AST nodes}} \times 100 \tag{3.10}$$

We then compute the overall CSS similarity score S_{CSS} where q is the number of matched CSS file pairs:

$$S_{\text{CSS}} = \frac{1}{q} \sum_{a=1}^{q} (S_{\text{CSS}}^{(a)}) \tag{3.11}$$

Visual rendering

Algorithm 6 Extract contour properties from screenshot

function FindContourProperties(s)
  $g \leftarrow$ ConvertToGrayscale(s) ▷ Convert screenshot to grayscale
  $C \leftarrow$ FindContours(g) ▷ Find contours from grayscale image
  $A_{\text{total}} \leftarrow \sum_{i=1}^{|C|}$ ContourArea(C_i) ▷ Calculate total contour area
  $M_{\text{total}} \leftarrow \sum_{i=1}^{|C|}$ ContourMoments(C_i) ▷ Calculate total contour moments
  $A \leftarrow \sum_{i=1}^{|C|} \left(\frac{\text{ContourArea}(C_i)}{A_{\text{total}}}\right)^2$ ▷ Calculate normalized area measure
  $M \leftarrow \sum_{i=1}^{|C|} \left(\frac{\text{ContourMoments}(C_i)}{M_{\text{total}}}\right)^2$ ▷ Calculate normalized moments measure
  return $(|C|, A, M)$ ▷ Return contour count, area, and moments

To compare the visual rendering, we propose a novel approach to compare screenshots of web pages $v'_{ij}(t)$ and $v'_{ik}(t)$ based on contour-based analysis using the Canny edge detection algorithm [29]. This method captures granular changes in visual content such as text changes, broken links, or missing images. Algorithm 6 describes the process of extracting contour properties from a screenshot. Using the contour properties $|C|$ (number of contours), A (weighted aggregate contour area), and M (weighted aggregate contour moments), we compute the geometric mean GM and finally, the similarity score S_{Visual}:

$$GM = \sqrt[3]{|C| \times A \times M} \tag{3.12}$$

$$S_{\text{Visual}} = \left(1 - \frac{|GM_1 - GM_2|}{\frac{GM_1 + GM_2}{2}}\right) \times 100 \tag{3.13}$$

Cookies

We analyze the cookies set by each web page, considering their names, values, and domains. The comparison involves matching cookies between $v'_{ij}(t)$ and $v'_{ik}(t)$ based on their names and domains. For matched cookies, we compute the similarity score S_{Cookies} where the "Number of matching cookies" counts cookies that are present in both web pages and have identical names and domains:

$$S_{\text{Cookies}} = \frac{\text{Number of matching cookies}}{\text{Total number of cookies}} \times 100 \tag{3.14}$$

HTTP requests

During the crawl of $v'_{ij}(t)$ and $v'_{ik}(t)$, we intercept HTTP requests and responses made during the crawl. We match requests based on exact URLs. Then we compute the similarity score S_{HTTP}:

$$S_{\text{HTTP}} = \frac{\text{Number of matching requests}}{\text{Total number of requests}} \times 100 \tag{3.15}$$

Textual Content

To compare the textual content of web pages $v'_{ij}(t)$ and $v'_{ik}(t)$, we use Diff Match Patch [91], a diff library based on Myer's diff algorithm [160]. This algorithm computes the Levenshtein distance d between two sequences of characters, which represents the minimum number of single-character edits required to change one word into the other. We compute the similarity score S_{Text} where $|T_{ij}|$ and $|T_{ik}|$ are the lengths of the textual content of the web pages $v'_{ij}(t)$ and $v'_{ik}(t)$:

$$S_{\text{Text}} = \left(1 - \frac{d}{\max(|T_{ij}|, |T_{ik}|)}\right) \times 100 \tag{3.16}$$

3.5.2 Computing the similarity score S_{score}

Algorithm 7 outlines the steps for computing the similarity score S_d of each dimension, after which we aggregate to obtain the overall similarity score S_{score} (cf. Equation 3.6) between web pages $v'_{ij}(t)$ and $v'_{ik}(t)$.

Algorithm 7 Compute similarity between two web pages

1: **function** ComputeSimilarity(w_i, t, u_j, u_k)
2: $v'_{ij}(t) \leftarrow V'(t)[w_i][u_j]$
3: $v'_{ik}(t) \leftarrow V'(t)[w_i][u_k]$
4: **for** each dimension d in $\{1, 2, \ldots, n\}$ **do**
5: $S_d \leftarrow$ ComputeDimensionSimilarity$(d, v'_{ij}(t), v'_{ik}(t))$
6: Compute overall similarity score:
7: $S_{\text{score}} \leftarrow \frac{1}{n} \sum_{d=1}^{n} (S_d)$
8: **return** $\{S_d\}_{d=1}^{n}, S_{\text{score}}$

3.5.3 Determining information relevance in UBE

From Equation 3.6, let $U = \{u_1, u_2, \ldots, u_m\}$ be the set of all UBE constructs used for the crawl. For a given URL w_i, we compute pairwise similarity scores between all crawls $r_{ij}(t)$

for each pair $u_j, u_k \in U$. This gives us a set of similarity scores across all dimensions and all pairs of UBE constructs:

$$\mathcal{S} = \{S_{\text{score}}(r_{ij}(t), r_{ik}(t)) \mid u_j, u_k \in U, t \in T\} \tag{3.17}$$

By iterating over all attributes $x_i \in u$, we determine the relevance of each attribute using $\delta(x_i)$ (cf. Equation 3.1). This leads to the construction of $I = \{x_i \in \Delta u \mid \delta(x_i) = 0\}$ the set of irrelevant attributes, and $R = \{x_i \in \Delta u \mid \delta(x_i) = 1\}$ The set of relevant attributes. Algorithm 8 outlines the steps to deduct the sets I and R hence defining a new UBE construct $u'' \subseteq u$ that balances functionality and web privacy.

Algorithm 8 Deriving relevant and irrelevant attributes from valid static crawl data

function DERIVEATTRIBUTES($U, D, V'(t), w_i$)
  $S_{\text{score}} \leftarrow$ empty list ▷ Store S_{score} across $V'(t)$
  $I \leftarrow$ empty set ▷ Initialize irrelevant attributes set
  $R \leftarrow$ empty set ▷ Initialize relevant attributes set
  for each pair of UBE constructs $u_j, u_k \in U$ **do**
    $v'_{ij}(t) \leftarrow V'(t)[w_i][u_j]$ ▷ Retrieve data from $V'(t)$ for $\{w_i, u_j\}$
    $v'_{ik}(t) \leftarrow V'(t)[w_i][u_k]$ ▷ Retrieve data from $V'(t)$ for $\{w_i, u_k\}$
    for each dimension $d \in D$ **do**
      $S_d \leftarrow$ COMPUTESIMILARITY($v'_{ij}(t)[d], v'_{ik}(t)[d]$) ▷ Compute S_d
    $S_{\text{score}}[u_j, u_k] \leftarrow \frac{1}{n}\sum_{d=1}^{n}(S_d)$ ▷ Aggregate S_{score}
  for each attribute $x_i \in \Delta u$ **do** ▷ Evaluate relevance of x_i in Δu
    $S_{\text{with}_{x_i}} \leftarrow S_{\text{score}}(v'_{ij}(t), v'_{ik}(t))$ ▷ Similarity across U (full UBE)
    $S_{\text{without}_{x_i}} \leftarrow S_{\text{score}}(v'_{ij}(t) \setminus \{x_i\}, v'_{ik}(t) \setminus \{x_i\})$ ▷ Similarity without x_i
    if $S_{\text{with}_{x_i}} - S_{\text{without}_{x_i}} \neq 0$ **then**
      $\delta(x_i) \leftarrow 1$ ▷ x_i is relevant
      ADD(x_i, R) ▷ Add to relevant attributes set
    else
      $\delta(x_i) \leftarrow 0$ ▷ x_i is irrelevant
      ADD(x_i, I) ▷ Add to irrelevant attributes set
  return I, R ▷ Return irrelevant and relevant attributes sets

3.6 Threats to validity

3.6.1 Bot detection

In Section 3.4, we addressed the validity of the crawl results. One reason these results could be invalid is that websites may detect our crawler and treat it as a bot. Some websites employ sophisticated techniques to detect bots and consequently change or restrict the content they serve. These techniques involve analyzing browser behaviors, HTTP headers, JavaScript execution environments, and network patterns [236, 144, 115, 108]. Simulating the UBE while avoiding bot detection is a challenging problem because mimicking real

user behavior in browsers can introduce inconsistencies, which in turn become indicators of automation and bot behavior [66, 96]. To mitigate the impact of bot detection on our UBE simulation, browser instrumentation tools like Playwright helps us control real browsers, thereby reducing the likelihood of detection. We also implement strategies to address basic bot detection mechanisms, such as removing automation indicators like the `navigator.webdriver` property in Listing 3.2. However, despite these measures, there remains a significant chance that some websites may still detect our crawler as a bot, especially when we begin experimenting with varying UBE constructs. Therefore, while our approach minimizes detection risks, it is possible that some of the data we consider valid is actually content intended for bots rather than humans.

3.6.2 Investigating web page functionality

Our study focuses on analyzing the JavaScript code itself rather than its runtime functionality. While our approach examines observable differences across multiple similarity dimensions, it does not determine whether these differences lead to functional changes. For example, assessing the similarity of JavaScript code without executing and interacting with the web page makes it difficult to understand the practical implications of any discrepancies. JavaScript events such as clicks and other user interactions might malfunction or exhibit performance issues that are not detectable through our analysis. A functional analysis could provide deeper insights into the changes resulting from our work [35, 5]. Moreover, functionality encompasses not only usability but also performance, accessibility, and alignment with user expectations. Crawlers lack the contextual understanding and experiential insight that humans possess. Therefore, our method does not address the nuances of functional degradation that may result from UBE modifications.

3.6.3 Scope of relevance of UBE attributes

Our method aims to identify irrelevant UBE attributes by observing the absence of differences in web page behavior when certain attributes are modified or omitted. However, the absence of observable changes does not conclusively prove irrelevance. Some attributes may influence aspects of the web page that are not captured by our similarity dimensions, such as server-side processing, analytics, or personalization that does not manifest in the similarity dimensions we investigated. Furthermore, the relevance of UBE attributes may vary across different contexts, user scenarios, or over time. A web page might not utilize a specific attribute under certain conditions but rely on it in others. Our snapshot-based analysis may not capture these conditional dependencies, potentially leading to incorrect conclusions about attribute relevance.

3.7 Conclusion

In this chapter, we presented a novel approach for determining the relevance of information in the UBE by simulating website visits using different UBE constructs. In this approach, we propose to use web similarity techniques to assess how restricting or omitting specific UBE attributes impacts website behavior. This method allows for the identification of irrelevant attributes that can be safely restricted, thereby enhancing user privacy without compromising usability.

We began by categorizing UBE information into geolocation, device, and browser attributes and formalized our objective. We then explored different ways to simulate the UBE. Through browser instrumentation, we elaborated a method to simulate varying UBE constructs and crawl the web to study the impact of UBE attributes. For the crawl process, we proposed a method to remove dynamic content and focus on static elements in our analysis. We introduced Similarity Radar, a multidimensional approach to compare web page behavior across similarity dimensions in the web page. This approach allows computing similarity scores through which the impact of UBE attributes can be assessed. These scores help determine irrelevant UBE attributes which can be restricted to enhance user privacy.

Our novel approach not only provides a systematic framework to analyze the relevance of information in the user's browsing environment but also lays the groundwork for further analysis. In the following chapter, we delve deeper into the impact of device information beyond determining nonessential information. Specifically, we will examine how our method can help determine the type and severity of changes in website behavior resulting from modifying device information.

Chapter 4

Exploring the impact of device information on the web

4.1 Overview

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 study, our goal was 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 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 showed 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 brought some proof that it may be time to turn the page on the UA string and retire it from current web browsers.

4.2 Motivation

In the early days of the web, web browsers had different technological stacks and would not interpret HTML tags the exact same way [95]. 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 [105], the UA string is still being used and its history is long, granular, and complex [125]. 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 [7]. In particular, UA exposed by browsers can be leveraged by browser fingerprinting, which has seen a steady rise in the past decade [132]. 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 [226], whose goal is to freeze the UA string as it is one of the most revealing information in fingerprints [62, 134].

In this study, we investigated 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 assessed this impact by comparing the similarities between web pages when browsed using standard browsers versus so-called None-browsers. None-browsers are the standard browsers from which we removed the User-Agent request-header field, the `navigator.userAgent`, and other identifying information in the JavaScript. For the comparison, we crawled 270,048 web pages from 11,252 domains and observed 100% similarity in the web pages before the execution of JavaScript, demonstrating the absence of differential serving. However, 8.4% of the web pages changed after the execution of JavaScript, hence highlighting the dependency on UA for content adaptation. We conducted a change impact analysis on UA-dependent web pages and found that third-party scripts from ads, bot detection, and content delivery network services were behind the changes in the web pages.

This study addressed the following research questions:

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

4.3 UA-Radar: Measuring Web Similarity in the Wild

4.3.1 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 4.1, 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

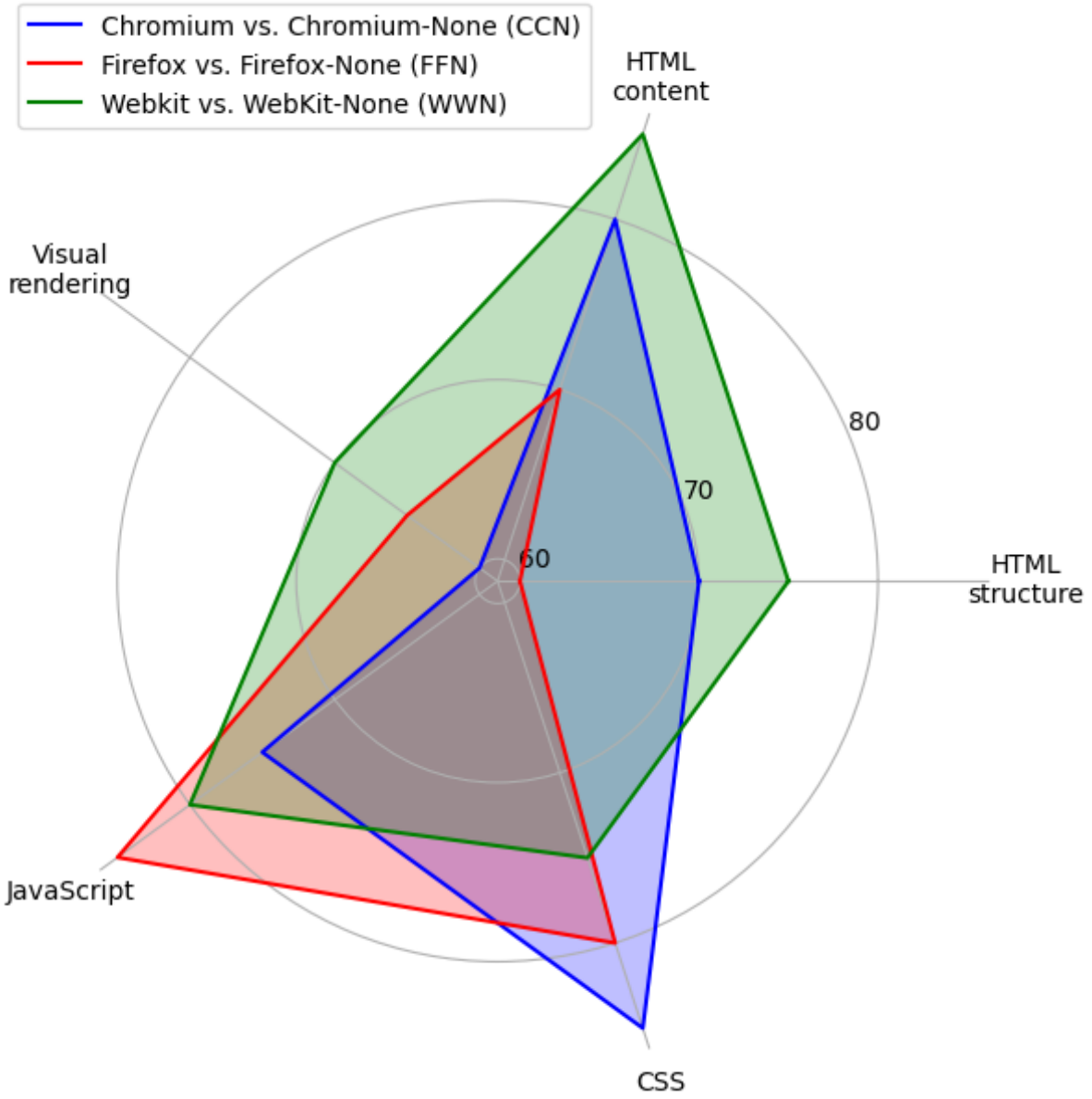


Figure 4.1: Similarity radar for a web page: the above represents the similarity between standard browsers and their None counterparts when accessing the home page of `www.academiabarilla.it`. Each colored pentagon corresponds to a single comparison, and its vertices represent the similarity scores across 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.

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 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. 4.2).

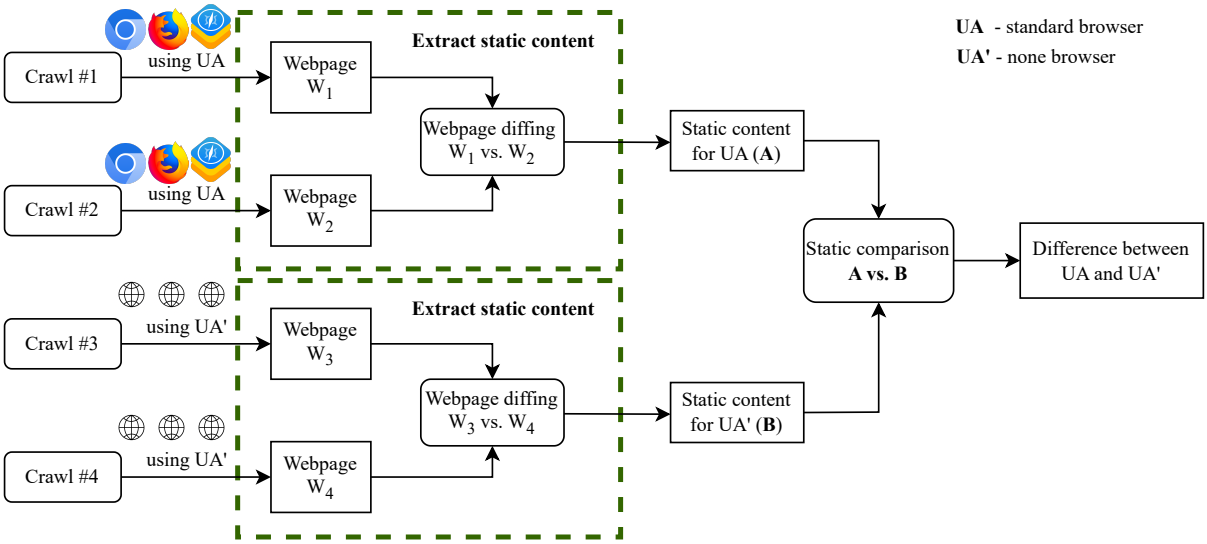


Figure 4.2: Highlighting web page similarity: standard browser (UA) versus None-browser (UA'). We crawl each web page twice using standard browsers (Chromium, Firefox, WebKit) and their None-browser counterparts. The dual crawl allows us to filter out dynamic content and focus on the static content of 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.

4.3.2 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 [18, 187, 25]. We use SFTM, a DOM tree matching tool, to measure the similarity of the HTML structure of two web pages in the wild [23]. 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 [91] that implements Myer's diff algoirthm [160] 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 [160]. 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 $S_2 = d/|E|$ [257].

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 [50]. 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 [73]. We

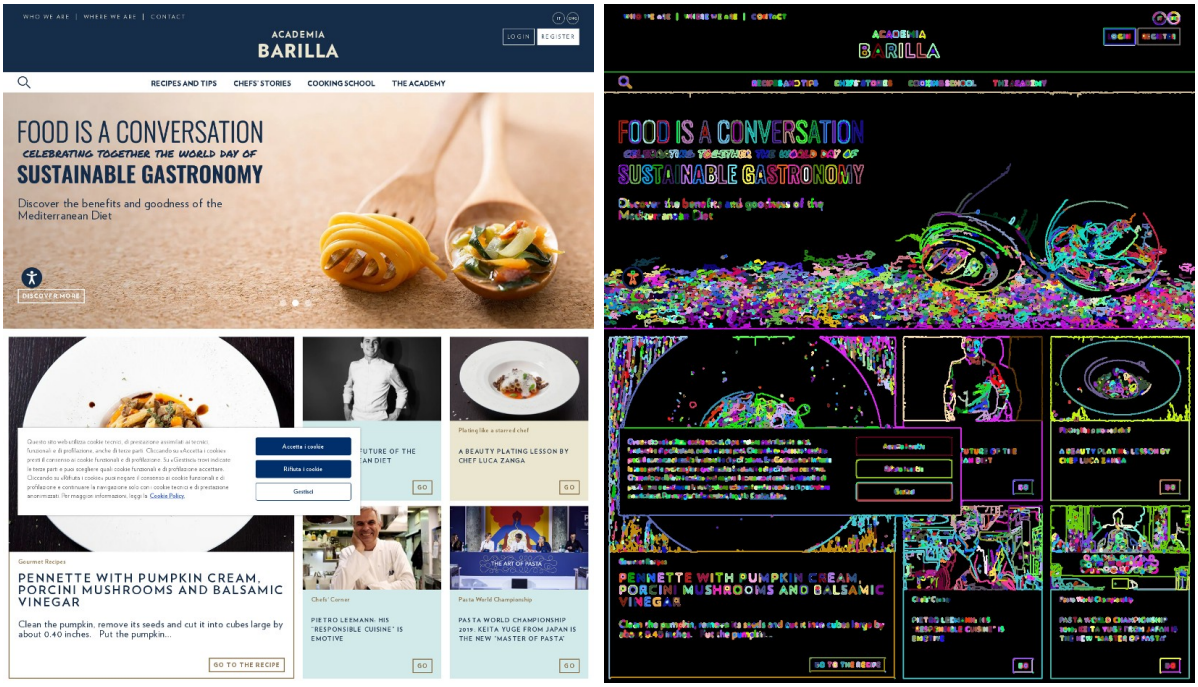


Figure 4.3: Contour-based visual analysis: the figure illustrates the process of contour-based analysis on a screenshot taken from `www.academiabarilla.it`. The top image represents the original screenshot, while the bottom image shows the identified contours (edges), representing different objects and 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.

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 $S_3 = |D|/|E|$.

Comparing visual rendering State-of-the-art image comparison algorithms are based on perceptual hashing, histogram, or by looking at pixel-by-pixel changes [60, 127, 206, 229, 241]. 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 [29]. Our algorithm computes the number of edges (contours) in a screenshot of the web pages, hence calling it contour-based analysis.

We rely on the OpenCV [21] library to retrieve contours from an image. OpenCV implements the shape analysis algorithm by Satoshi *et al.* [215]. 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 9 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 9 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) ▷ Convert image s to grayscale image g
3: C = cv::findContours(g) ▷ Find contours C from image g
4: Y = cv::contourArea(C) ▷ Store area of each contour in Y
5: Z = cv::boundingRect(C) ▷ Bounding rectangle Z
6: Let $A = 0$, $M = 0$ ▷ 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: **return** <|C|, A, M>

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

$$GM = \sqrt[3]{|C| \times A \times M} \tag{4.1}$$

$$S = \frac{|GM_1 - GM_2|}{(GM_1 + GM_2)/2} \tag{4.2}$$

4.4 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.

4.4.1 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) [174]. 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 JavaScripts 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 Appendices. 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 [1]. We chose the Tranco list as its ranking of website popularity surpasses other sources of web traffic analysis [175]. Nevertheless, previous studies have expressed concerns about the methodology used to create popular lists, such as Tranco and their representativeness [199]. 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[10]. 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 Appendices.

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 4.1 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 4.1: 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**

4.4.2 Empirical Results & Findings

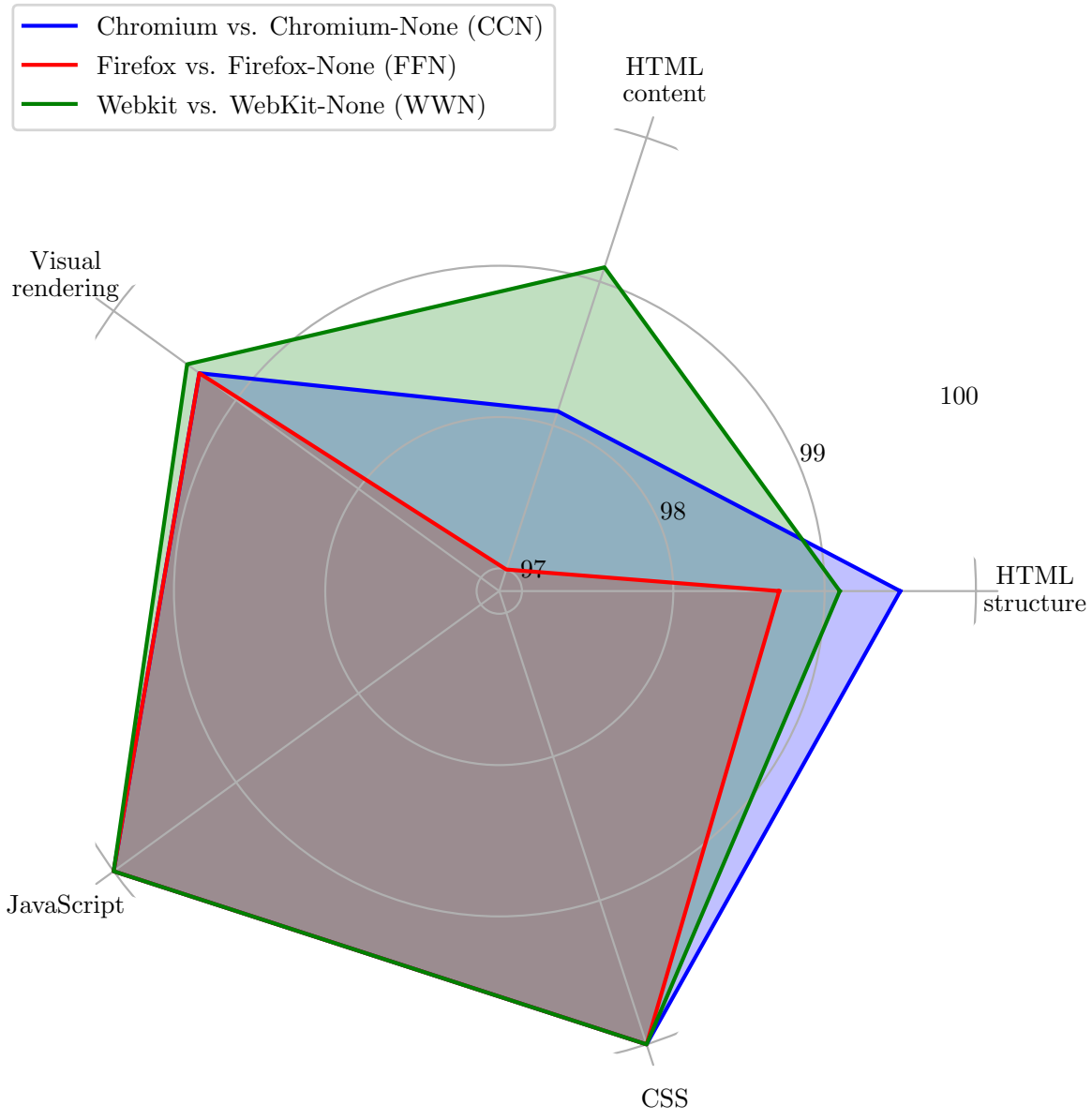
In this section, we use the following notations: CCN for the comparison between pages from Chromium against Chromium-None, FFN for Firefox against Firefox-None and WWN for WebKit against WebKit-None.

Differential Serving

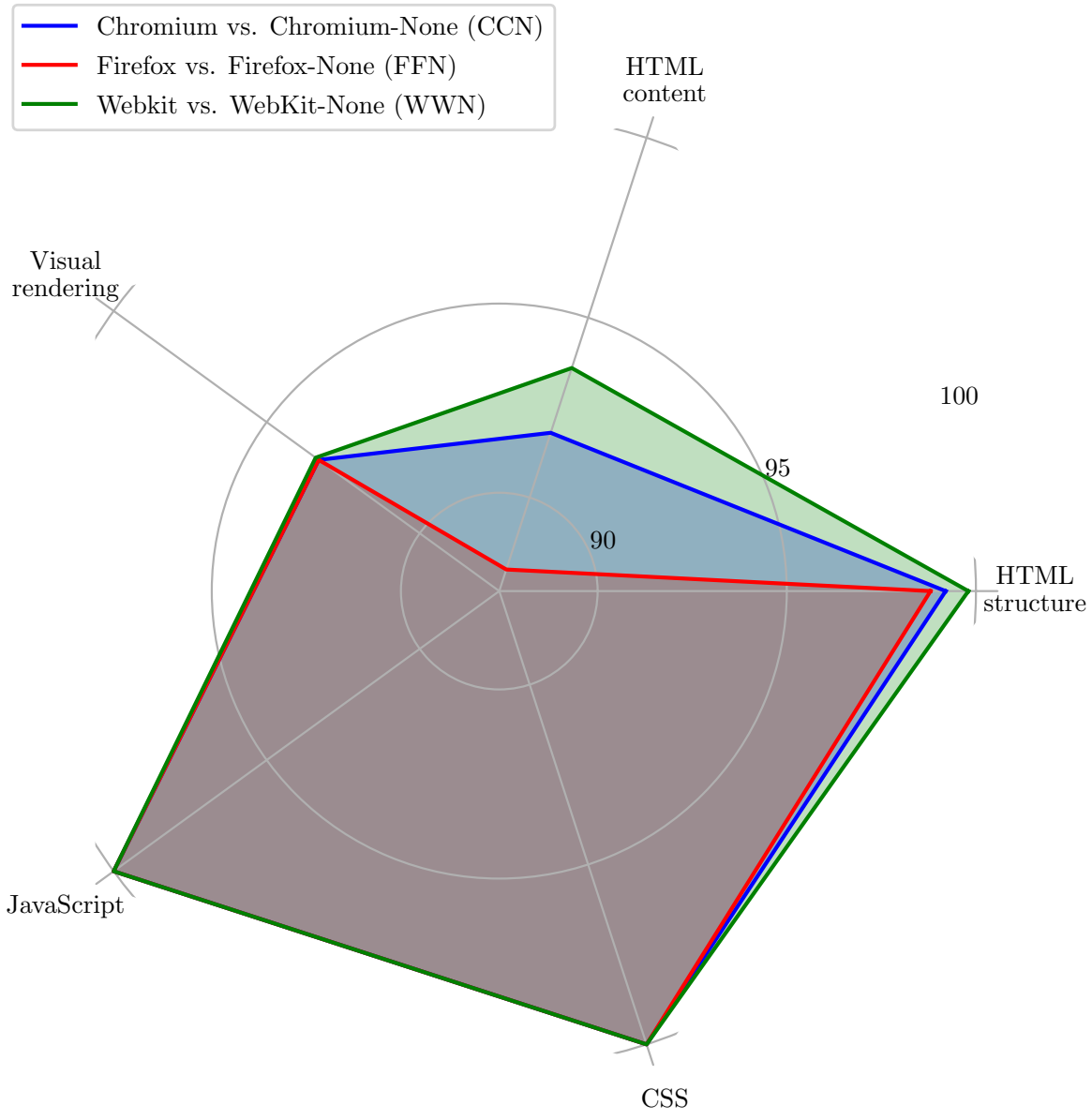
The average similarity scores before JS is executed are 100% for all the tested browsers (CCN, FFN, and WWN) on HTML structure, HTML content, JS, and CSS (cf. Figure 4.4a). One takeaway is that web servers reply to all HTTP requests with the same HTML document, regardless of the fact that the UA in the 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

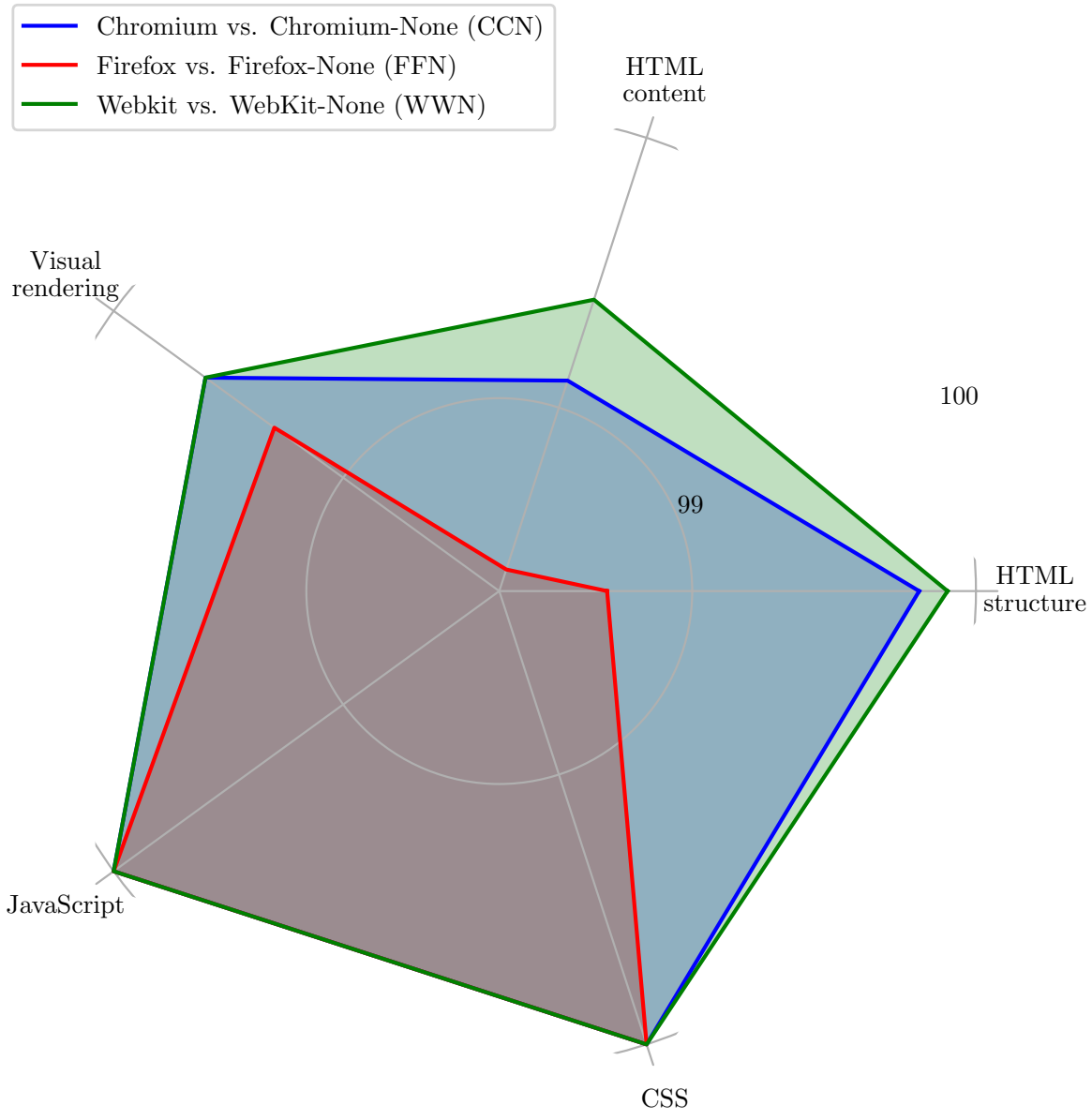
When JavaScript is executed, the average similarity scores remain 100% on JS and CSS. However, there are changes in visual rendering, the HTML structure, and the HTML content. 158 out of the 11,252 domains are not 100% visually similar for at least one of


Chromium vs. Chromium-None (CCN)
Firefox vs. Firefox-None (FFN)
Webkit vs. WebKit-None (WWN)
HTML content
Visual rendering
100
99
98
97
HTML structure
JavaScript
CSS

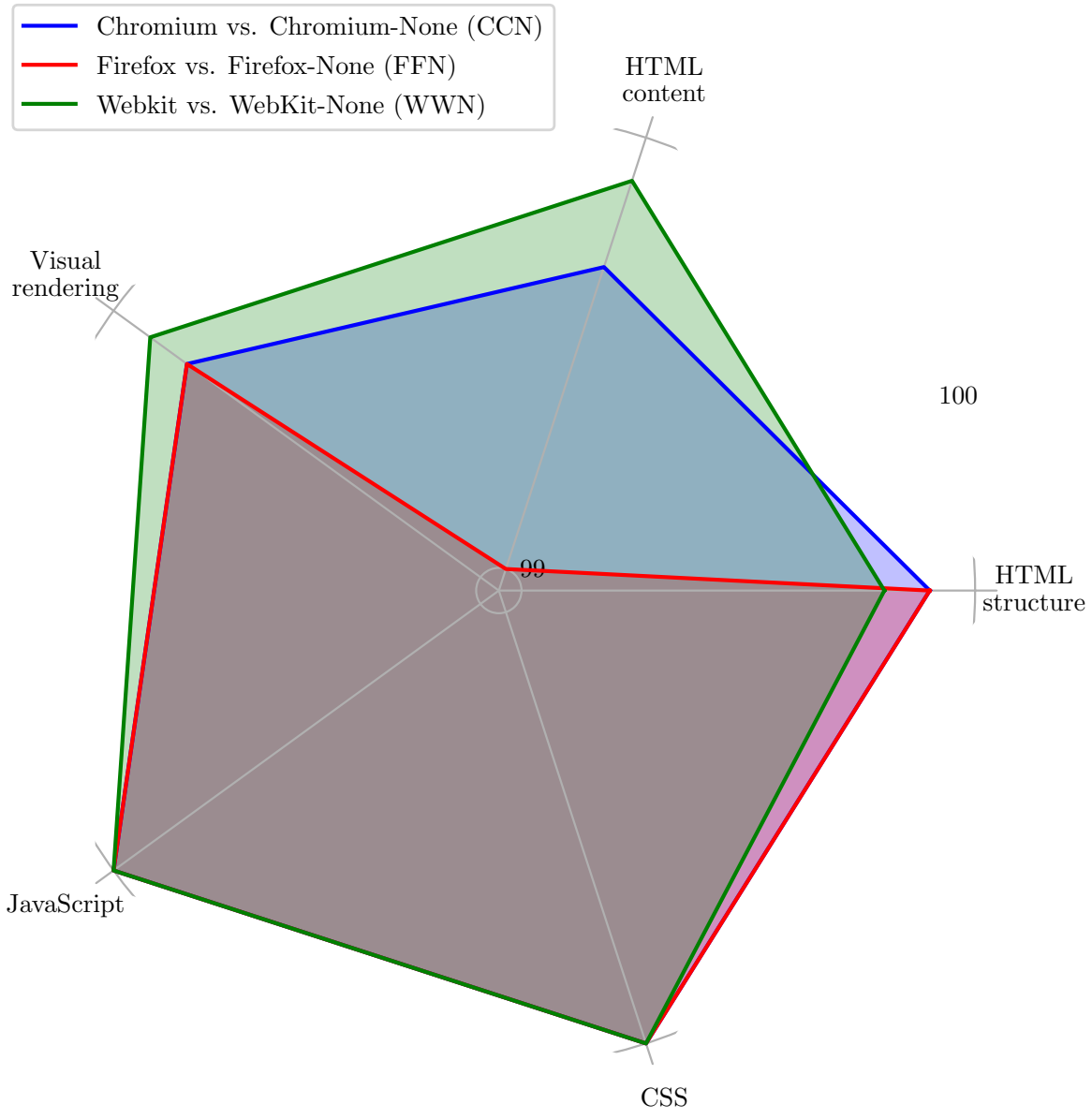
(a) All


Chromium vs. Chromium-None (CCN)
Firefox vs. Firefox-None (FFN)
Webkit vs. WebKit-None (WWN)
HTML content
Visual rendering
100
95
90
HTML structure
JavaScript
CSS

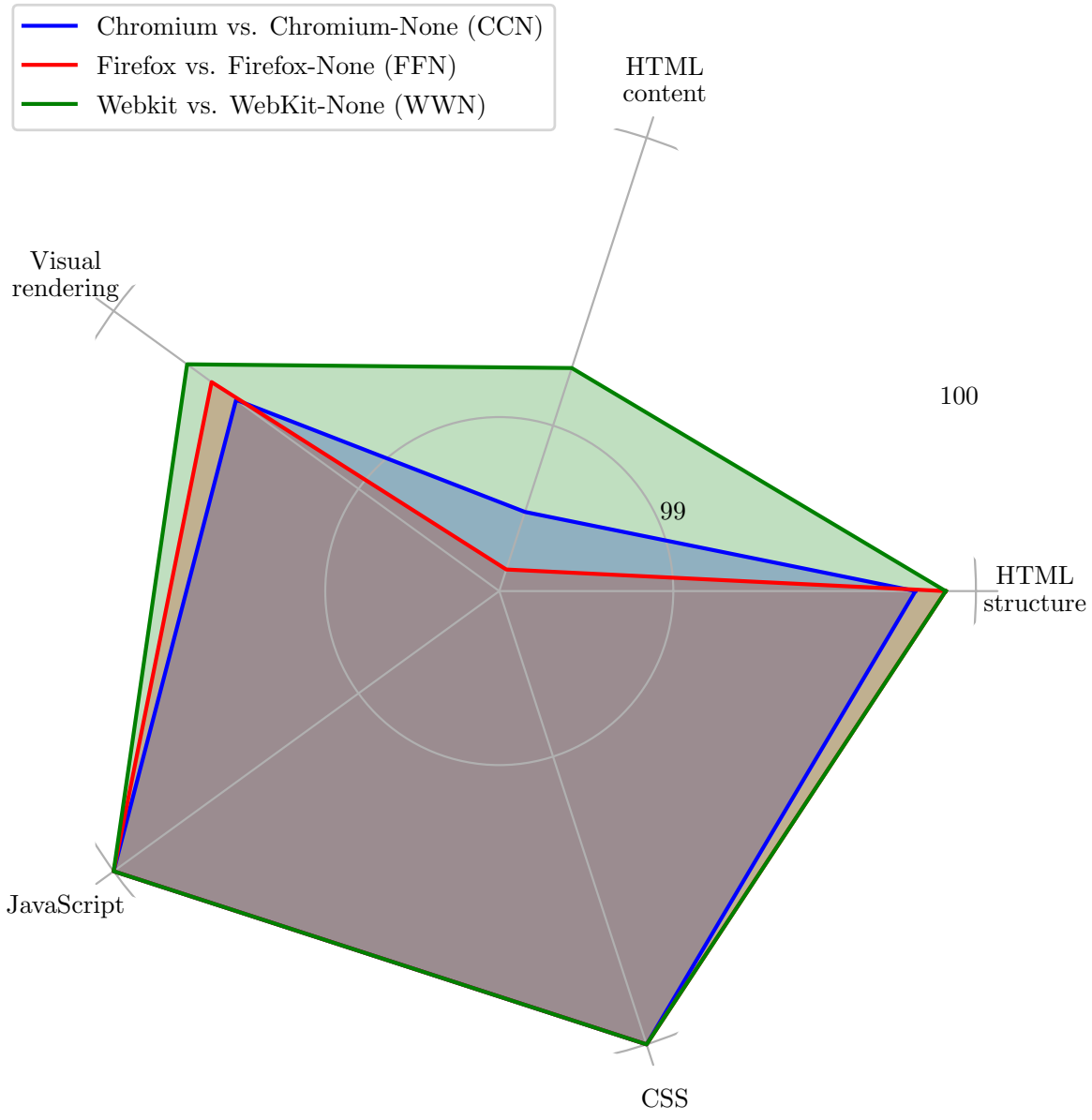
(a) News



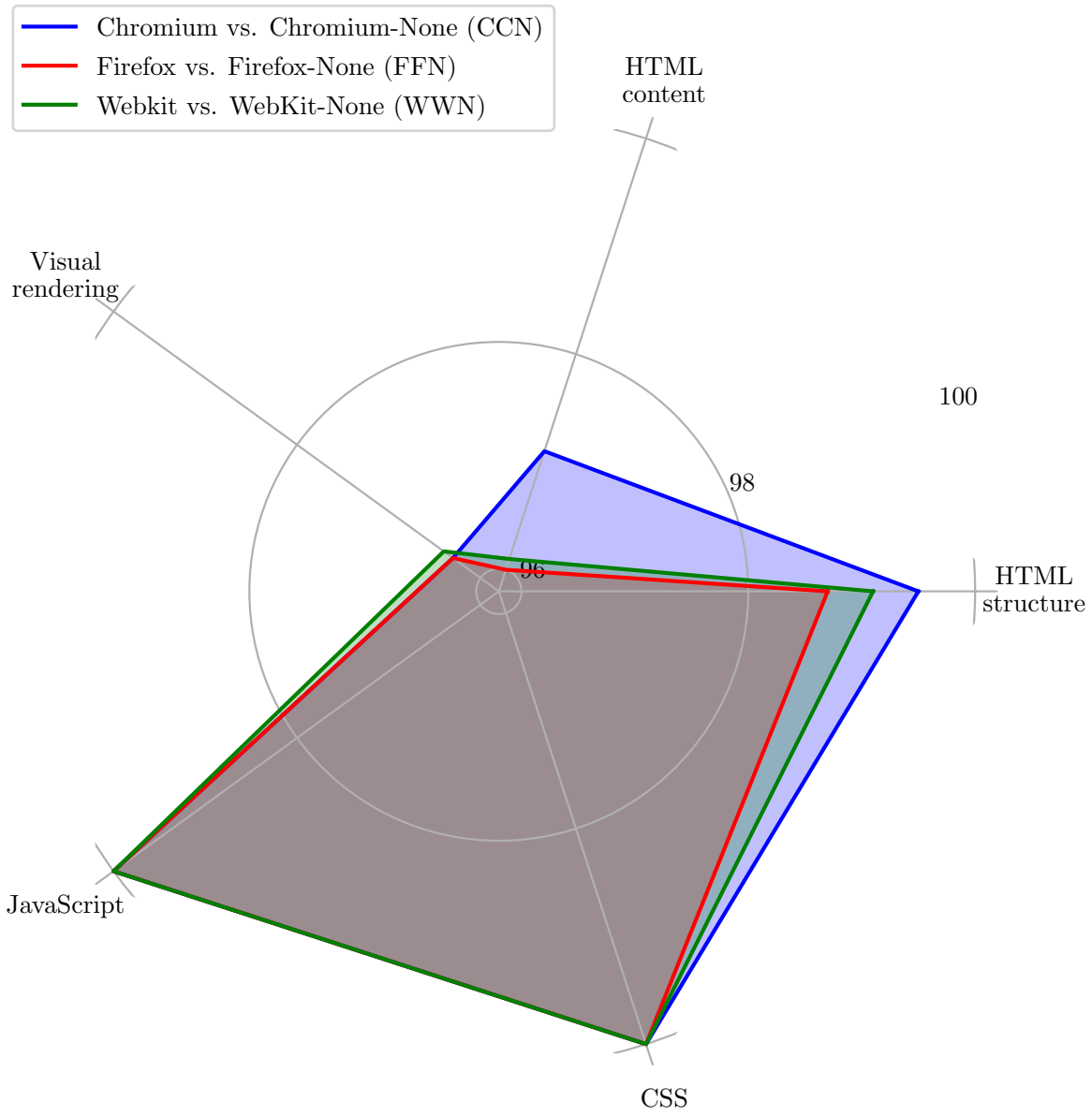
(b) Internet Services


Chromium vs. Chromium-None (CCN)
Firefox vs. Firefox-None (FFN)
Webkit vs. WebKit-None (WWN)
HTML content
Visual rendering
100
99
HTML structure
JavaScript
CSS

(c) Business


Chromium vs. Chromium-None (CCN)
Firefox vs. Firefox-None (FFN)
Webkit vs. WebKit-None (WWN)
HTML content
Visual rendering
100
99
HTML structure
JavaScript
CSS

(d) Marketing



(e) Online Shopping

Figure 4.5: 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.

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 4.2 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 [33].

Table 4.2: 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 4.3: 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 4.5 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 4.5a), 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 4.5b, 4.5c, and 4.5d), 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 4.5e), 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

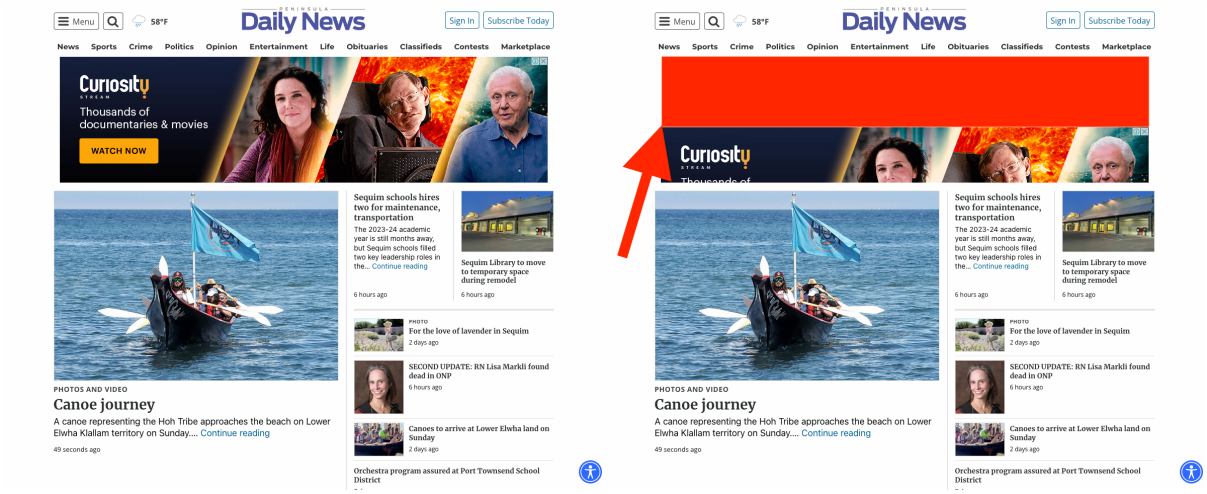
Due to the intensive manual efforts required to analyze the changes, out of the 955 domains that changed because of the None-browser, we manually analyzed the changes for 204 domains. We detected 10 patterns of the impact of those changes and used the 10 patterns to apply heuristics to the rest of the 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.* [191]. Algorithm 10 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.

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",

Algorithm 10 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 4.2) 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 ΔCCN ← StaticComparison(C, CN)
3:     Let ΔFFN ← StaticComparison(F, FN)
4:     Let ΔWWN ← StaticComparison(W, WN)
5:     Let ΔCF ← StaticComparison(C, F)
6:     Let ΔCW ← StaticComparison(C, W)
7:     Let ΔFW ← StaticComparison(F, W)
8:     if ΔCCN = ΔFFN & ΔFFN = ΔWWN then
9:         Let R ← [ΔCCN, ΔFFN, ΔWWN]
10:        Let N ← [ΔCF, ΔCW, ΔFW]
11:        if margin-{top,bottom} ∈ ΔCCN then
12:            return"Margin collapsing fail"
13:        else if white-space: wrap ∈ ΔCCN then
14:            return"Soft-wrap fail"
15:        else if page-break-before,after ∈ ΔCCN then
16:            return"Unnecessary blank  lines"
17:        else if <tag css> ∈ ΔCCN then
18:            return"Inline css  changes"
19:        else if <img src> ∈ ΔCCN then
20:            return"Lazy loading fail"
21:        else if <iframe width|height> ∈ ΔCCN then
22:            return"Displaced iframe"
23:        else if <tag inactive|disabled> ∈ ΔCCN then
24:            return"Disabled component"
25:        else if CAPTCHA|403|error ∈ ΔCCN then
26:            return"Browser not identified"
27:        else if R ≠ N then
28:            return"Content restriction"
29:        else
30:            return"No pattern"
31:    else
32:        Let x ← ΔCCN ≠ ΔFFN ≠ ΔWWN
33:        Let y ← ΔCF ≠ ΔCW ≠ ΔFW
34:        if x & y then
35:            return"No impact"
36:        else
37:            return"No pattern"
```



(a) Standard browser - normal margin rendering (b) None browser - failure of margin collapse

Figure 4.6: 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.

"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.

Listing 4.1: Example of unintentional restriction: a function that relies on known UA (parameter b) in a script from Google Ad Manager.

```
function rn(a, b, c, d) {
  O(a.K, {
    transition: c / 1E3 + "s",
    "transition-timing-function": d,
    "margin-top": b
  })
}
```

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.

Table 4.4: 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**	**Occurences**
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

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 4.4 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 4.3 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 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 4.6, 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 4.1, 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.

(a) Standard browser - normal page access

(b) None browser - access intentionally restricted

Figure 4.7: Example of "unusable" problem severity: access to the web page is intentionally restricted when using a None Browser.

Listing 4.2: Example of intentional restriction: a function that runs a CAPTCHA test when the UA is not recognized.

```
function Hl() {
  var a, b;
```

```
  return "function" === typeof(null == (a = E.navigator) ? void 0 : null == (b
      = a.userAgentData) ? void 0 : b.getHighEntropyValues)
}
...
Hl() ? (d(), t(r.linkAttribution)) : r.enableRecaptcha && p("require", "
    recaptcha", "recaptcha.js");
```

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 4.7, 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 4.2, 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 4.8, 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'.

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

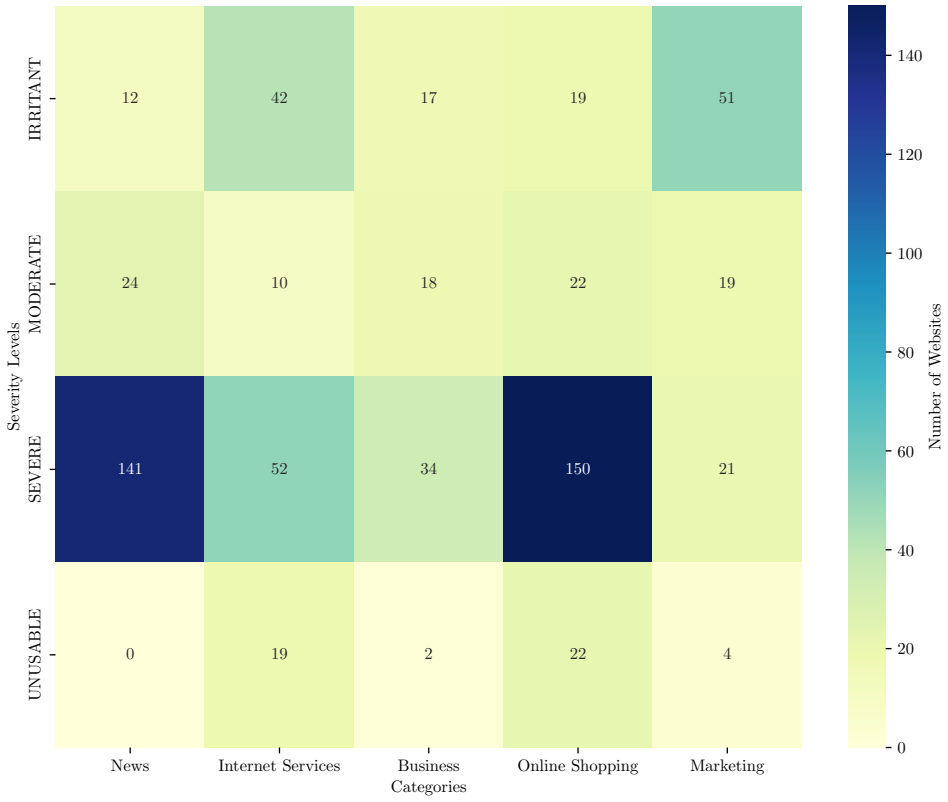


Figure 4.8: Problem severity distribution across website categories: this heat map depicts how changes in the UA impact different website categories, highlighting the prevalence of problem severity levels in each category.

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.

4.5 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 [109]. As mentioned in Section 4.6, 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.

4.6 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 [67]. 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 [164]. Cross-referencing our dataset with a list of known trackers from uBlock Origin (uBO) [103], we found that 612 domains (5.4%) out of the 11,252 were on the uBO list. 38.3%

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

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 [109]. Moreover, 584 domains return a Vary header that indicates 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.

4.7 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.

4.8 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.

Chapter 5

Exploring the geolocation impact on enforcing privacy policies

5.1 Overview

Regulatory frameworks, such as GDPR in Europe, CCPA in the United States, and LGPD in Brazil have made cookie consent banners a standard feature on websites, seeking user consent for cookies and tracking technologies. However, the visibility and impact of these banners may vary based on the user's geographic location. In our study, we introduce a novel automated visual detection technique to explore the enforcement of cookie consent banners. This methodology outperformed earlier techniques that depended on manual observations or inspection of HTML/CSS elements. Our analysis of $70,390$ web pages visited across five countries on different continents revealed geographical disparities in banner visibility, with France exhibiting the highest prevalence at 69% and Japan the lowest at 27%. This variation in visibility correlates with the state of cookies and tracking. Our further analysis of $351,950$ web pages indicated that third-party cookies increased post-banner interactions, notably in the US. Conversely, France, with its stringent regulations, maintained the lowest number of third-party cookies. Tracking trends closely followed these patterns, highlighting the relationship between banner visibility and user tracking. Our study underscores the significance of our new automated visual detection method, given that banner visibility has a direct correlation with tracking activities, highlighting its necessity for accurate insights into web privacy and user consent practices.

5.2 Motivation

The advent of the *General Data Protection Regulation* (GDPR) in Europe and the *California Consumer Privacy Act* (CCPA) in the United States has led to a significant increase in the prevalence of cookie consent banners on websites. These banners, which request the user consent for the use of cookies and other tracking technologies, have

become a ubiquitous part of the online experience [93]. However, the visibility and the impact of these banners can vary significantly depending on the geographic location of the user [230].

The methodologies in previous studies on the prevalence of cookie consent banners have relied on manual observations of small numbers of websites [153, 15, 97], or the inspection of HTML/CSS elements to detect those banners at scale [116, 119, 167, 19, 186, 198, 228]. There is a question of whether the presence of those elements on the web page guarantees the visibility of consent banners. In some cases, the banner is disabled if the user is visiting the website from a non-regulated area. Another challenge is the limited use of languages to understand the content of the banner to declare it a cookie consent banner. We conduct automatic visual detection of banners to investigate the visibility of consent banners in five countries from different continents. To address the limitation of languages we use automatic translation to decide on the type of the banner. We, furthermore, examine the geographical impact of those consent banners on the state of cookies and tracking in those countries. Our study sheds light on the following research questions:

- How does the visibility of cookie consent banners vary across continents?
- How does the state of cookies get impacted by regional variation in banner visibility?
- How does the state of tracking get impacted by regional variation in banner visibility?

To examine the variability in cookie consent banner visibility across continents and its impact on cookies and tracking, we introduced a novel technique for automated visual detection of these banners. With automatic translation, we addressed language limitations in identifying banner types. We further crawled across $14,078$ domains from five countries on different continents, under various scenarios, to answer the above research questions. Our newly introduced automated visual detection technique has proven to be considerably more effective than previous methods for identifying cookie consent banners. Traditional techniques, such as inspecting HTML/CSS elements or relying on manual observations, often face challenges. They might not accurately detect banners, especially when these are disabled in non-regulated regions. For instance, past research using these traditional methodologies reported a 47% prevalence of banners in Europe [186]. In contrast, our visual approach detected a prevalence of 69% in France. Such differences highlight the precision of our methodology. Moreover, our in-depth exploration across various web pages offers insights into the state of cookies and tracking, emphasizing the relationship between banner visibility and user tracking practices.

In the following sections, we will discuss our methodology, present our data collection and findings, and discuss their implications. We will also present the threats to the validity of our approach before concluding the study.

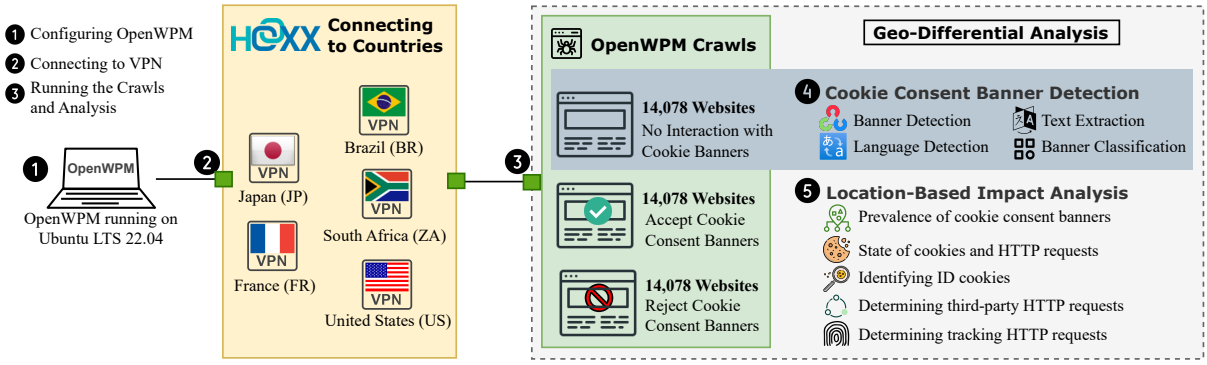


Figure 5.1: Overview of our methodology for analyzing the prevalence of cookie consent banners and the state of cookies and tracking. The process begins with the preparation of the crawler, ensuring accurate data collection and integration of browser extensions to accept or reject cookie consent banners. This is followed by establishing a geo-location connection using `HOXX VPN`. The crawler is then executed in three distinct instances: without interaction with the banner, with `IDCAC` accepting consent banners, and with `Consent-o-Matic` rejecting them. Screenshots are captured during the first instance for subsequent visual detection of banners. The final stages involve identifying websites displaying the banners and analyzing the state of cookies and tracking.

5.3 Web Crawling Methodology

In our empirical exploration of the enforcement of cookie consent banners, we employed our novel automated visual detection methodology to examine geographical disparities. Selecting countries was pivotal; our aim was to contrast regions with stringent data protection regulations, like those under GDPR and CCPA, against areas without such comprehensive regulations. While we aspired to cover more than five geographically diverse countries, we faced specific challenges. Reliable VPN services, essential for authentic geo-location representation, were sparse, leading to South Africa being our sole representation from Africa. Additionally, our constrained computing resources posed challenges for extensive crawls across numerous countries. Given these considerations, we selected Brazil, Japan, and South Africa – each distinctive and not under comprehensive regulations like GDPR or CCPA. We also incorporated France, impacted by GDPR, and the United States, impacted by CCPA. After determining the countries for our study, we conducted our research through a structured five-stage process. First, we prepared our crawler for efficient data collection and chose a reliable VPN service to maintain authentic connections to target countries. With this foundation, we ran the crawler across countries in three distinct scenarios: pre-interaction with the consent banner (`No Interaction`), and post-interaction (`accept`, and `reject`). We reran the post-interaction scenarios to facilitate the identification of ID cookies. During the post-interaction scenario, screenshots were captured to be used in the visual detection of banners.

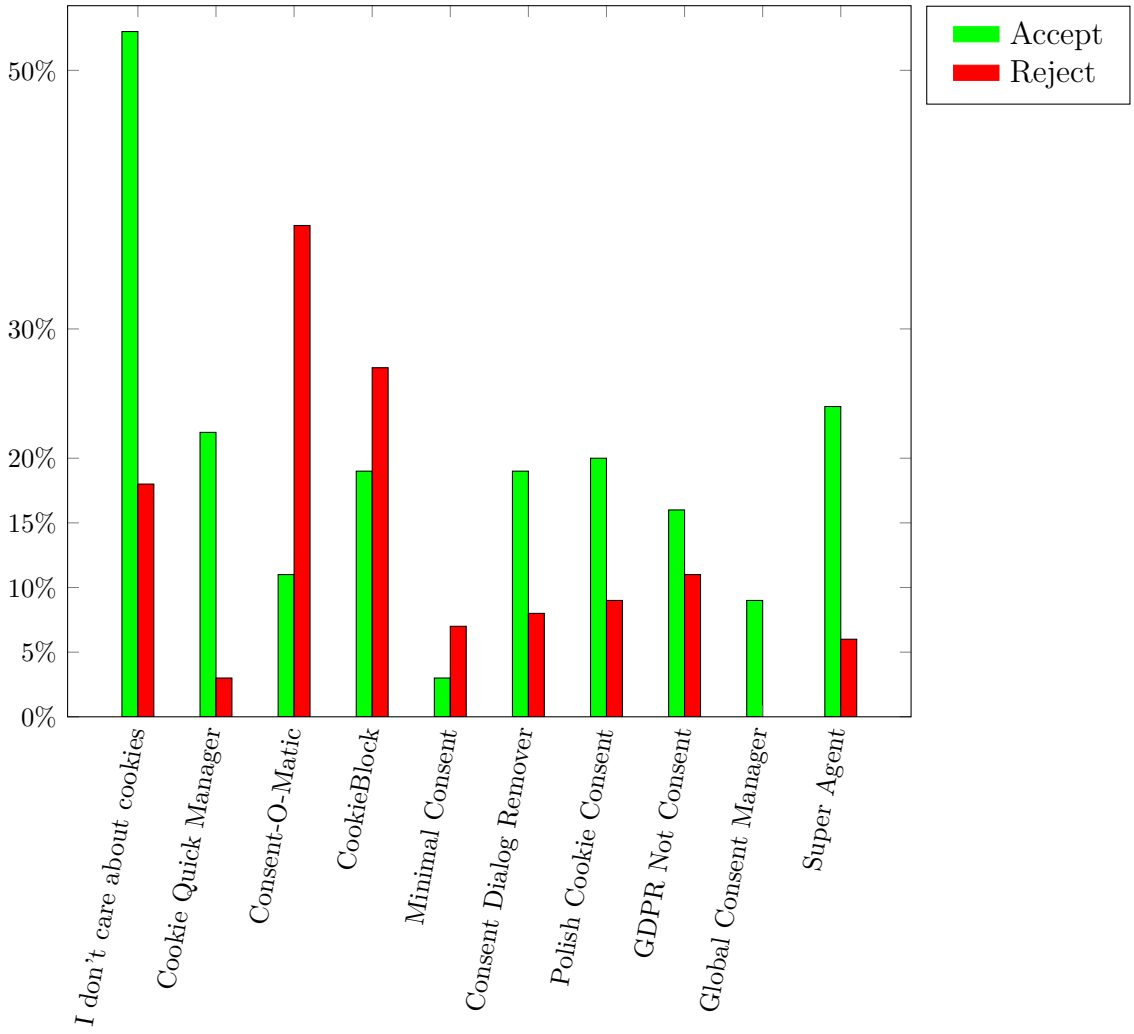


Figure 5.2: A comparison of the success rate of interaction, through either acceptance or rejection of cookies, by different Firefox browser extensions. The results are based on a sample experiment conducted on 100 domains, aiming to decide on suitable extensions for our analysis of the state of cookies and tracking, which required not removing cookies or blocking HTTP requests.

Country	Approach	Domains with banners	Banner Prevalence
BR	visual	4,932	35%
	`z-index`	3,788	27%
FR	visual	9,723	69%
	`z-index`	7,467	53%
JP	visual	3,801	27%
	`z-index`	2,919	21%
ZA	visual	5,849	43%
	`z-index`	4,492	33%
US	visual	4,435	32%
	`z-index`	3,406	25%

Table 5.1: Comparison of the number of domains displaying cookie consent banners and their corresponding prevalence percentages in five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US), as identified by two different detection methods - our automated visual approach and the `z-index` approach.

5.3.1 Preparing the crawler

Our web crawler is based on OpenWPM, a web privacy measurement framework designed to collect data for privacy studies [68]. OpenWPM utilizes Mozilla Firefox as its browser for operation. Within Firefox, an OpenWPM extension is installed to facilitate the collection of web measurements. We explored different approaches to enhance our crawler's capabilities, particularly in interacting with cookie consent banners. BannerClick [185], introduced by Rasaii *et al.* in their study [186], appeared to be a potential fit. However, integrating it without disrupting our existing pipeline proved challenging due to the specific browser profile it uses. Achieving compatibility would require having access to the identical browser profile and the precise browser version used in their study, necessitating the exploration of alternate avenues. While the CookieBlock study discusses the ability to engage with banners dynamically, the code provided falls short of implementing that [19].

Having not found a solution in recent studies, we searched for capable extensions in Firefox Browser Add-ons. Our search criteria included extensions that did not remove cookies or block HTTP requests, as we would need that data for analyzing the state of cookies and tracking. The top 10 extensions we found were: I don't care about cookies (`IDCAC`) [107], Cookie Quick Manager [44], Consent-O-Matic [43], CookieBlock [45], Minimal Consent [157], Google Consent Dialog Remover [90], Polish Cookie Consent [177], GDPR Not Consent [83], Global Consent Manager [86], and Super Agent [214]. We collected a sample of 100 domains to experiment with those extensions and choose one to use for our data collection process. We manually visited the web pages of those 100 domains to observe when cookie consent banners are displayed. We then manually interacted with the banners and observed the cookies set in the `accept` and `reject` scenarios and, then, built a ground truth data set to assess the performance of the chosen extensions to test. Furthermore, we used a crawler to run a Firefox browser with different

profiles, each with a separate extension among the chosen Top 10, and an extension we built to track the interaction with the cookie banners. Finally, we compared the cookies set by each extension in the `accept` and `reject` scenarios against our ground truth data set. Figure 5.2 shows the success rate of interaction (`accept` or `reject`) by each of those extensions. `IDCAC` accepted banners the most (53%) but Consent-O-Matic rejected those banners the most (38%). Note how CookieBlock is close to Consent-O-Matic without the use of the feature extractor. CookieBlock could have done better had the code been available. Eventually, we had to use two extensions instead of one, `IDCAC` for accepting cookie consent banners and Consent-O-Matic for rejecting them.

Despite these challenges, our methodology accounted for the reality of cookie consent banners on the Internet. There are instances where websites impose a minimum set of functional cookies that cannot be rejected [117]. Furthermore, dark patterns in cookie consent interfaces often make it difficult to fully reject cookies or obfuscate the process of doing so [204, 102]. In our approach, using `IDCAC` to accept cookies and Consent-O-Matic to reject them helped us interact better with the banners during our study. However, we recognize that this approach does not capture every detail that our visual approach can. Also, building a new tool based on our visual detection of cookie consent banners was outside the scope of this study due to time and resource constraints.

5.3.2 Connecting to countries

For our analysis, we needed to connect our crawlers to the countries we targeted to collect authentic measurements. We could use *virtual private network* (VPN) services or run our crawlers in different cloud computing regions. Due to resource constraints, we chose the avenue of VPN services. We considered NordVPN [166], HOXX VPN [104], and ProtonVPN [178]. Our choice of those three was influenced by several factors. First, their extensive server coverage ensures a wide geographical spread, which is crucial for our study. Second, their proven stability and speed are essential to maintain the integrity of our data collection process [255, 30, 232]. Lastly, previous research endeavors have highlighted their reliability in ensuring genuine localized browsing experiences, making them ideal for our study [150, 222, 258]. While NordVPN had worked initially, they eventually suspended our account because their terms of service do not allow web scrapping. Concerned that ProtonVPN shares almost similar terms with NordVPN and could lead us to the same outcome, we opted for HOXX VPN and contacted them to let them be aware of our crawl scenarios. The adoption of HOXX VPN ensured that websites presented our crawlers with content tailored to users from the target geographic locations. This helped us understand the variability of cookie consent banners across different regions.

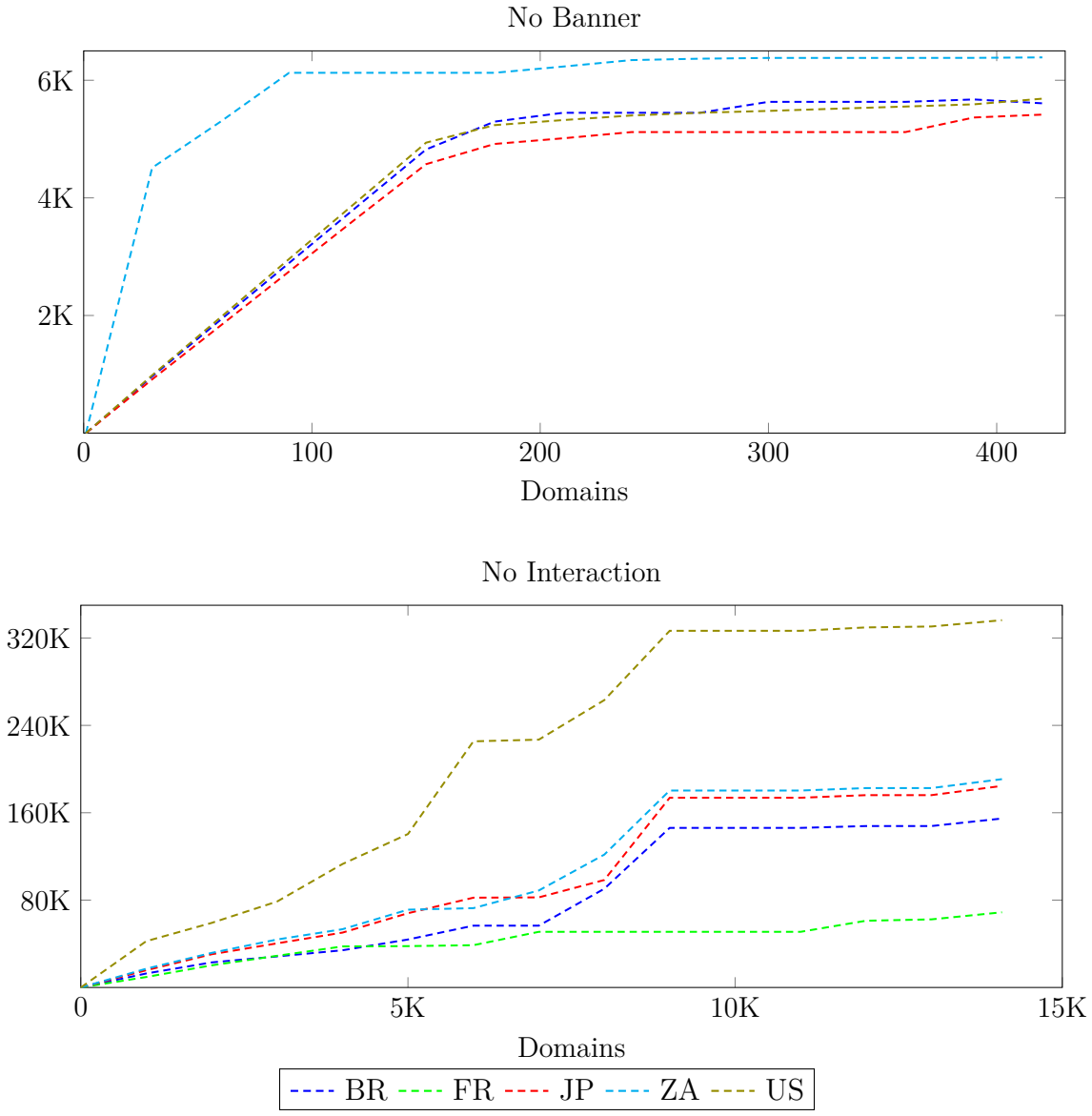


Figure 5.3: Distribution of third-party cookies across websites for different countries. The upper plot illustrates the number of third-party cookies served to the user when no banner is shown and the plot at the bottom illustrates when the user has not interacted with the consent banner yet.

5.3.3 Visual detection of banners

To detect cookie consent banners, we used different tools and approaches. First, we used the screenshots captured using OpenWPM in the crawl without interaction with the banners. Then, we used OpenCV [21], an open-source computer vision library, to detect the presence of these banners. Our process began with transforming the color screenshot into a grayscale image, simplifying the image by removing color data and allowing us to concentrate on structural details. We further simplified the image by converting the grayscale image into a binary format. It is important to note that not all banners create an obscure background or significantly alter pixel intensity. Therefore, instead of solely

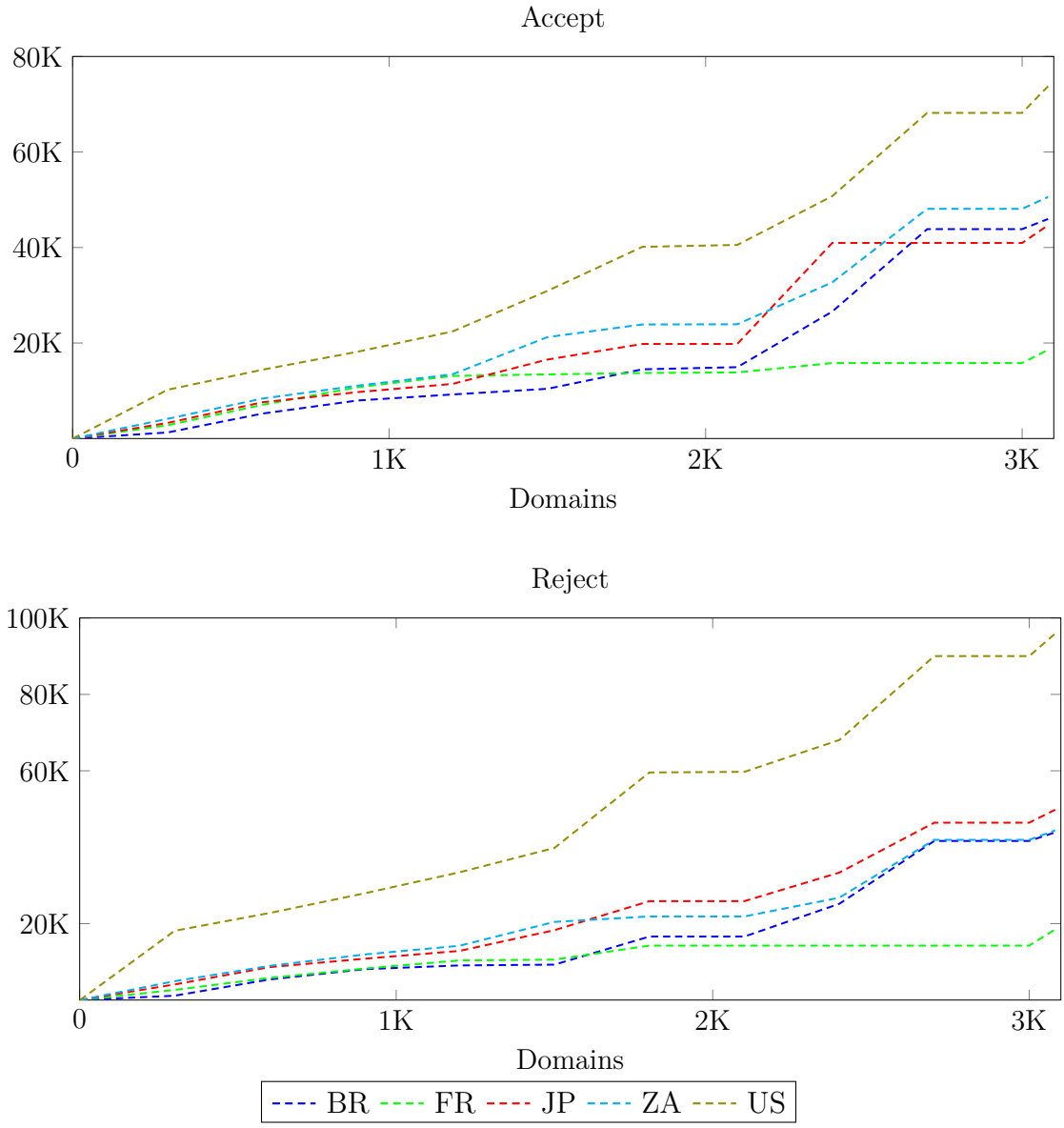


Figure 5.4: Distribution of third-party cookies across websites for different countries. The upper plot illustrates the number of third-party cookies after the user accepts cookies, and the plot at the bottom illustrates the situation when the user rejects consent to cookies.

relying on pixel intensity to distinguish between banners and the background, we based our analysis on geometric attributes. Utilizing the binary image, we performed a contour analysis to identify potential banners, focusing on rectangular shapes [216]. This is because cookie consent banners commonly present themselves as rectangles, regardless of their background. We also considered the relative size and position of these rectangles. This approach helped us to detect rectangle or four-sided objects included in the web page. While some of the detected rectangle objects could be a cookie consent banner, at this point we could not reliably determine that as a fact, so we wanted to make a distinction between a cookie consent banner and any other rectangle object. Soe *et al.* determined that the typical height of a banner is at least one-third of the browser screen height [203].

It is difficult to determine a threshold for the height of a typical banner, so we assumed that each rectangle object detected from the screenshot is a potential banner.

We used Tesseract, an *optical character recognition* (OCR) engine that extracts text from images [218], to determine the textual content within the pixel coordinates of identified banners. As the banners were detected from the binary images where text details could be lost, we applied the coordinates back to the original screenshots when using Tesseract. Since we were crawling from different geo-locations with various languages, it was necessary to ensure that Tesseract knew the language of the text in the banner. This step was important as the language used on the web page can vary based on the user geolocation, and specifying that language to Tesseract improved accuracy in extracting text from the banners. For that, we used Google Cloud Translation API [40], to detect the language used on the crawled websites and then specify those languages when running Tesseract. After that, we had groups of text, each deemed to be a banner and we wanted to determine whether any of those banners was a cookie consent banner. Therefore, we proceeded to the classification of the banners.

5.3.4 Classification of banners

One of the techniques used by previous studies to determine a cookie consent banner was to match pre-selected words against the text from web pages [186]. While this can have limited success in the case of common languages, it becomes a problem when the language on the crawled website is unknown. For that, we have decided to use the same dataset as discussed previously, of detected languages on the websites to classify the banners. Our goal was to isolate cookie consent banners from other types of banners. To achieve this, we employed the *Term Frequency-Inverse Document Frequency* (TF-IDF) model to transform text from each banner into a vector representation that we can use in clustering techniques [184, 147]. The TF-IDF model measures the importance of a term within the banner's text while offsetting its overall prevalence across all banners, accounting for the frequent appearance of certain terms.

We used the k-means clustering algorithm on banners after transforming them with TF-IDF [202]. This algorithm divides banners into groups, with each banner being part of a group that is similarly close. To decide the best number of groups, we looked at the silhouette score, which tells us how similar an item is to others in its group compared to items in other groups. We ran the k-means clustering nine times, with group numbers from 2 to 10. We chose this range because it gave us a good balance with regards to computations. We picked the group number that had the highest silhouette score. After grouping, we looked at which groups had the most banners. We then found the average TF-IDF vector for each of these groups and picked out the top three words. These words give a quick idea of what the banners in that group are about. For example, if the top words are *use*, *cookie*, and *accept*, we can guess that the group is about cookie

consent banners. We adjusted our method if we came across banners that were different or unique. Language was also important. For non-English banners, we used the Google Cloud Translation API to translate the top three words. This helped us to group banners correctly, even if they were in different languages. We have listed all the top three words we used in the appendix.

5.4 Analysis

After setting up our pipeline, it was time to collect data and conduct our analysis on the prevalence of consent banners and the state of cookies and tracking. We ran thirty crawls in total, six for each country that we studied namely, Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US). The six crawls per country consisted of two crawls without interaction with the cookie consent banners, two crawls where we used `IDCAC` to accept the cookie consent banners, and finally two crawls where Consent-O-Martic rejected them. The dual crawl in the `no interaction`, `accept`, and `reject` scenarios in each of the five countries permitted us to understand the state of cookies and tracking by looking at the same page and the same country.

5.4.1 Data collection

We used the Tranco list to choose the domains to crawl [1]. We chose the Tranco list as its ranking of website popularity surpasses other sources of web traffic analysis [140]. We also chose Tranco due to the fact that it integrates other rankings, such as the Chrome User Experience Report [48] and Cloudflare Radar [41] and the fact that it was possible to configure local Tranco rankings for our target countries [194]. 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 [10]. This was not a concern for our study as our objective was to assess the prevalence of cookie consent banners and their impact on cookies and tracking, and home pages are by default where those banners are triggered.

Due to the need to conduct a uniform, yet representative, study taking into account the impact of cookie consent banners on local domains, we configured local Tranco top 10k lists from BR, FR, JP, ZA, and the US. We combined those lists to make a single list of 50,000 domains, from which we eventually got a list of 14,078 unique domains. We crawled the homepage of each of those domains six times from each country, collecting twice the data for scenarios when (1) the user does not interact with the consent banner, (2) the user accepts the consent banner, and (3) the user rejects the consent banner. In total, we crawled the homepages of 14,078 domains thirty times.

5.4.2 Prevalence of cookie consent banners

We ran benchmarks comparing our automated visual approach for detecting cookie consent banners on the selected set of 14,078 domains against the `z-index` approach. In the `z-index` approach, we examined layered components on a web page for potential cookie consent banners by checking for elements with higher `z-index` values, which often indicate overlaying elements like banners. We further incorporated text matching, searching specifically for the words we had obtained, using the banner classification process, within these elements. By contrasting the `z-index` and text matching method against our automated visual approach, we aimed to assess the effectiveness of each technique in identifying cookie consent banners. Table 5.1 summarizes the results of the comparison of each method in detecting cookie consent banners. The discrepancy between the two approaches can be attributed to the limited capability of the `z-index` approach to comprehend the complexity of the code used to display the banner.

The results of our automated visual approach in detecting cookie consent banners show a variation in the prevalence of the banners. Out of the crawled 14,078 websites, France had the highest prevalence of consent banners at 69%, more than twice that of the United States, which had 32%, while Japan had the lowest prevalence at 27%. While France and Japan sit on both ends of the spectrum, we observed the lower ends occupied by South Africa at 43%, Brazil at 35%, and the United States. The high prevalence of cookie consent banners in France is due to the strict data protection regulations enforced in the European Union (EU). The EU's GDPR and ePrivacy Directive necessitate explicit user consent for the use of cookies on websites, explaining the high percentage of domains with banners [239, 260]. Previous GDPR fines have also contributed to the higher prevalence of cookie consent banners in France [193].

Despite Japan's introduction of the Amended Act on the Protection of Personal Information, the exact scope and enforcement level of this legislation remains unclear, possibly contributing to the lower prevalence of cookie consent banners observed. Unlike the well-established and strictly enforced data protection regulations in the EU, Japan's regulatory framework may not yet exert a strong influence on the practices of website operators regarding cookie consent. This situation might explain the 27% prevalence of domains displaying cookie consent banners in Japan, as compared to the 69% prevalence in France. The uncertainty surrounding the enforcement and the breadth of the Amended Act could lead to less adoption of cookie consent banners in Japan. This is also the case for the US which does not have a federal law regulating the use of cookies, but state-level laws like the California Consumer Privacy Act (CCPA) and Virginia Consumer Data Protection Act (CDPA) consider cookies as personal information [32, 74]. While this may contribute to an elevated prevalence of consent banners on websites operating or having a significant user base in these states, it remains unclear how those state-level laws can be enforced for a wider impact.

A slightly higher prevalence of cookie consent banners in South Africa compared to Brazil, the US, and Japan, may be explained by the *Protection of Personal Information* (POPI) Act that applies to businesses operating within South Africa [52]. POPI mandates data protection measures including cookie consent banners. While the enforcement clarity may not be entirely apparent, it instills a level of caution among website developers to avoid potential legal repercussions. The presence of cookie and personal data protection guidelines under Brazil's LGPD is acknowledged, yet effective compliance and enforcement have been scrutinized in several studies [71, 72, 28]. This might be a contributing factor to the observed low prevalence of cookie consent banners in Brazil. The regulatory landscape in the five countries significantly influences the prevalence of cookie consent banners. Effectiveness in stringency and enforcement drives higher prevalence while the opposite drives down the prevalence. These results underscore the importance of understanding regional nuances in cookie policies across countries and their impact.

5.4.3 State of cookies

The state of cookies on the crawled websites provides insights into the user experience when visiting those websites from different countries. Unlike first-party cookies, which are created by the domain a user is visiting, third-party cookies are set by a domain other than the one the user is currently visiting. These cookies are primarily utilized for tracking and online advertising purposes, enabling delivery of content to users based on their behavior or preferences [169, 208]. For instance, while visiting a news website, a user may be served an advertisement through a third-party domain, such as `ad.doubleclick.net`. This third-party domain can leave a cookie due to its embedded content on the original site, like an ad or tracking pixel.

Our results, as shown in Figure 5.3 indicate that before interaction, users in the US are served with the highest number of third-party cookies, more than double the number of other countries such as Brazil, Japan, and South Africa. Users in France are served the least number of third-party cookies. The high number of third-party cookies in the US may be explained by the fact that online advertising is often the business model of websites operating in or targeting users from the US [244]. As we have previously mentioned, this often means including third-party cookies to enable targeted advertising. In France, in addition to GDPR, the low amount of third-party cookies may be explained by the oversight of third-party cookies and other trackers by the country's data protection authority, CNIL [46]. Given this oversight, websites that operate in France or target French users tend to adopt a more conservative approach [240, 161]. Specifically, they may limit their use of third-party cookies to ensure that they remain compliant with local regulations and avoid potential penalties [77]. This proactive compliance can explain the low amount of third-party cookies served to users in France compared to other countries.

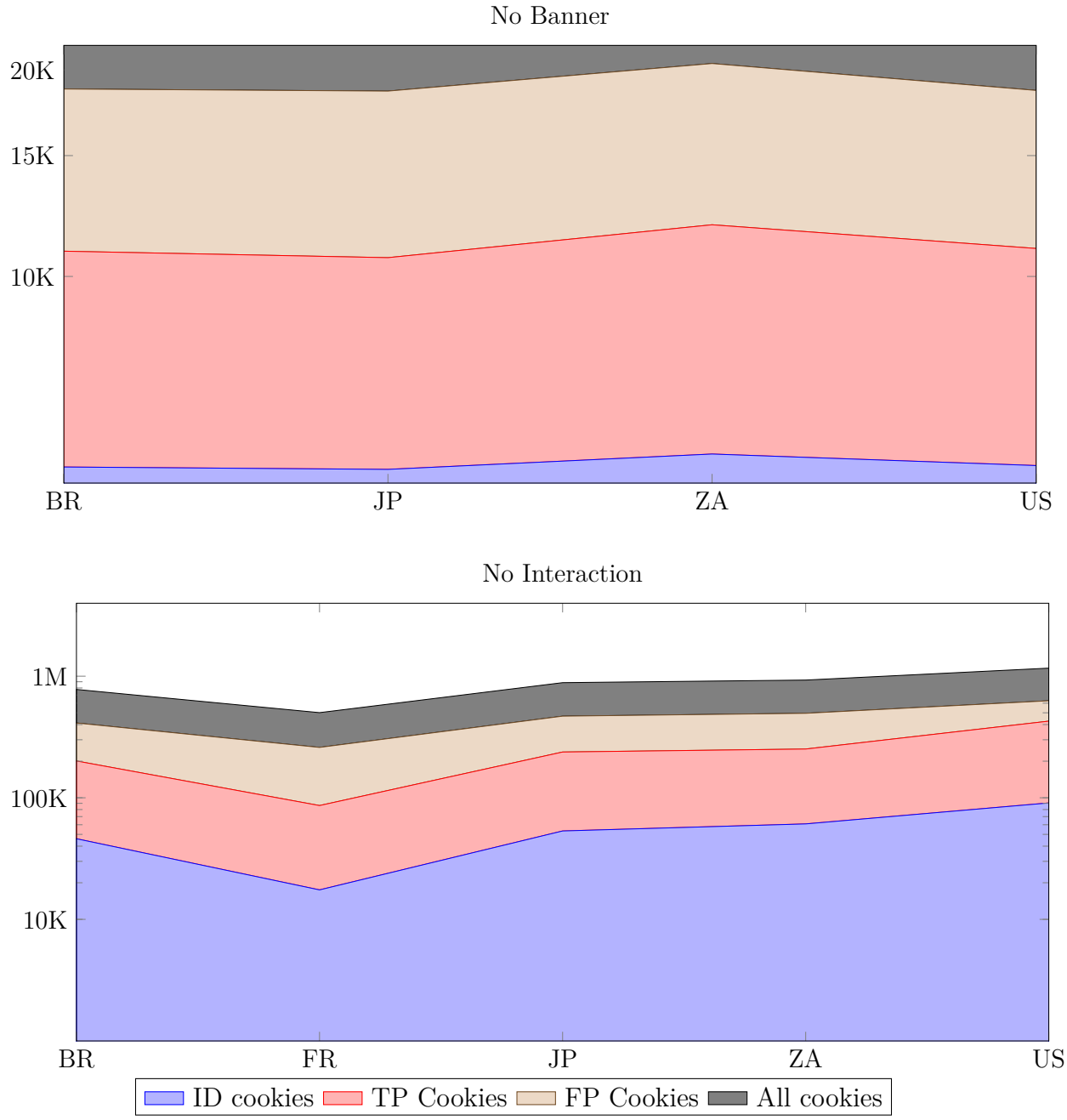


Figure 5.5: Comparative analysis of cookies set during the crawl across five countries under the `No Banner` and `No Interaaction` scenarios.

To avoid bias during the analysis of the state of cookies after interaction with consent banners, it was necessary to determine common websites where the extensions, `IDCAC` and `Consent-O-Matic`, had worked to accept and reject the banners. Out of 14,078 domains that we crawled, the extensions had both worked on 3,082 domains (21.9%). Thus, our further analysis of the state of cookies after interaction with banners focused on those 3,082 websites. Most of the disparity in the distribution of third-party cookies remains after interaction with consent banners, both in the *accept* and *reject* cases, especially in the case of the US having the highest number of third-party cookies being served and

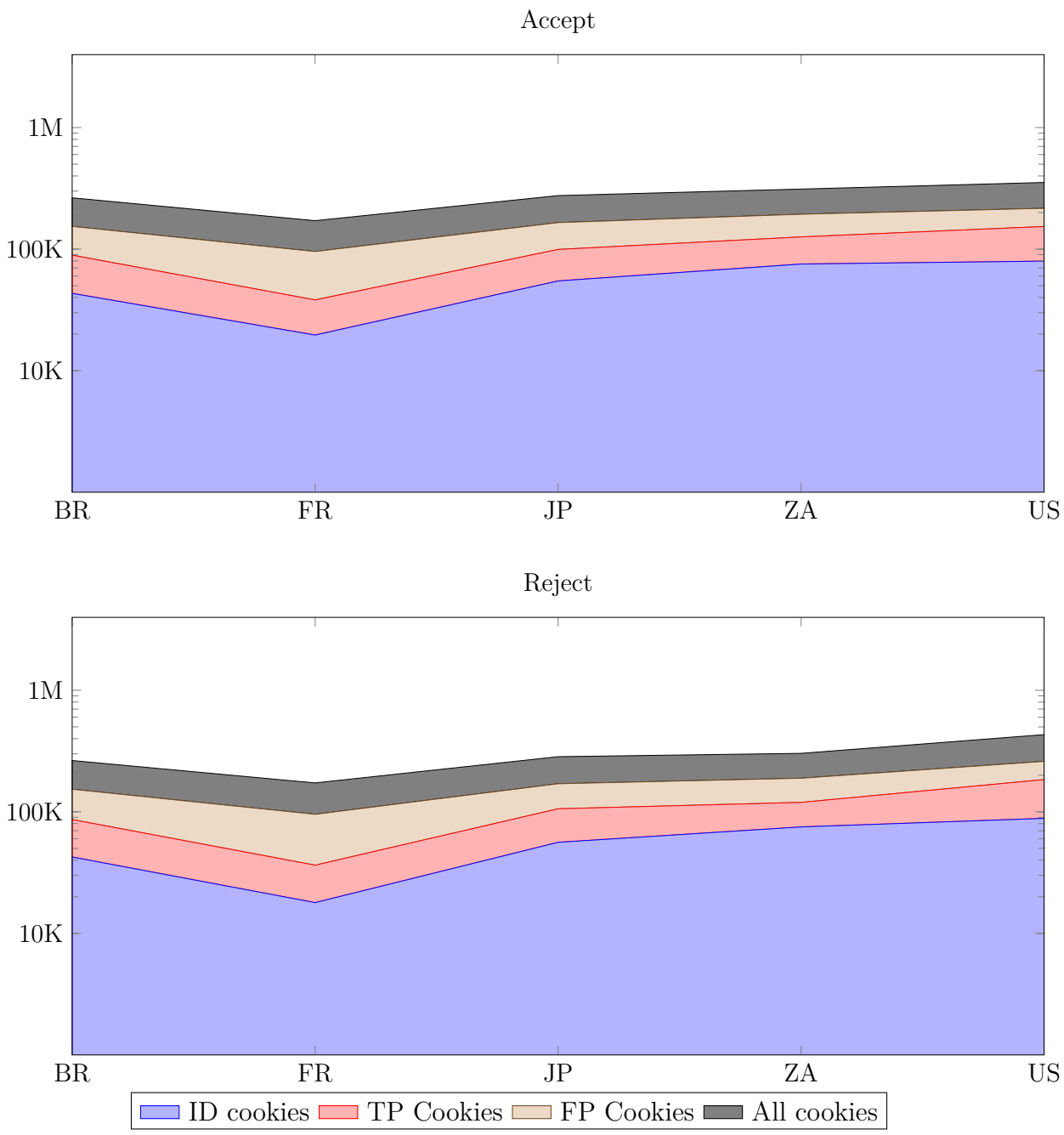


Figure 5.6: Comparative analysis of cookies set during the crawl across five countries under the `Accept` and `Reject` scenarios.

France having the least. It is worth noting, however, that although the overall number remains the lowest, third-party cookies increase in France when the user interacts with the consent banner. This is not the case in other countries. Figures 5.5 and 5.6 illustrates the state of cookies across countries before and after interacting with consent banners. The number of third-party cookies remains almost the same after user interaction with the consent banner, despite the fact that it is measured with a far smaller dataset. That indicates a surge in third-party cookies after user interaction with the content banner. This

prevalence of third-party cookies shows an interest by the website operators in tracking user behaviors across different websites.

This trend, along with the reliance on granular data by tracking mechanisms, led us to the topic of ID cookies. ID cookies are special because they are designed to provide distinct identifiers that can be used for tracking, especially for advertisers and data analytics platforms seeking to understand user behavior and preferences. To understand the prevalence of these ID cookies in our study, we used the methodology introduced by Wesselkamp *et al.* [251]. Compared to the vast number of cookies shown in Figures 5.5 and 5.6, ID cookies appear to be less prevalent. However, as shown by Figure 5.8, when users interact with consent banners (`accept` and `reject`), the number of ID cookies triples in each country except South Africa where it quadruples. In all cases, the number of ID cookies in the `Reject` scenario remain a bit less than that in the `Accept` scenario. That is the same case with websites serving those ID cookies. That increase can be linked to the use of dark patterns in the design of interfaces for cookie consent banners. Dark patterns trick users into making decisions they might not necessarily want to make, such as placing the "Accept All" button in a more prominent position or misleading users into thinking they have rejected cookies when they have not.

No banner The observations above raise the question: what happens when there is no cookie consent banner? Figure 5.7 illustrates the state of cookies in the `No Interaction` plot. However, the scenario of `No Interaction` differs from the situation where no banner is shown. To examine that later situation, we checked for websites where consent banners were accepted and rejected in France but no banner was shown on those websites in any other country. We found 429 such websites. The `No Banner` plot in Figure 5.5 illustrates the distribution of third-party cookies on those websites. Contrary to the trend in other plots where the US has the highest number of third-party cookies, half of these 409 websites show an unusually high number of third-party cookies with South Africa being on top. Figure 5.7 also corroborates this trend where South Africa has the highest prevalence of ID cookies and websites that serve ID cookies when no consent banner was shown. This suggests that consent banners present restraint to website operators who only unleash the majority of tracking when the user interacts with the banner. On the contrary, in regions where the regulations or their enforcement is not clear, users are targeted by a high number of cookies and tracking.

5.4.4 State of tracking

Figures 5.9 and 5.10 illustrate the analysis of HTTP requests during our crawls across the five countries. The substantial volume of requests can be attributed to essential HTTP requests that websites need to function properly. In general, when a website displays the consent banner, HTTP requests tend to have a similar pattern whether it is third-party or tracking requests except for the US. Before interaction with the consent banner, our

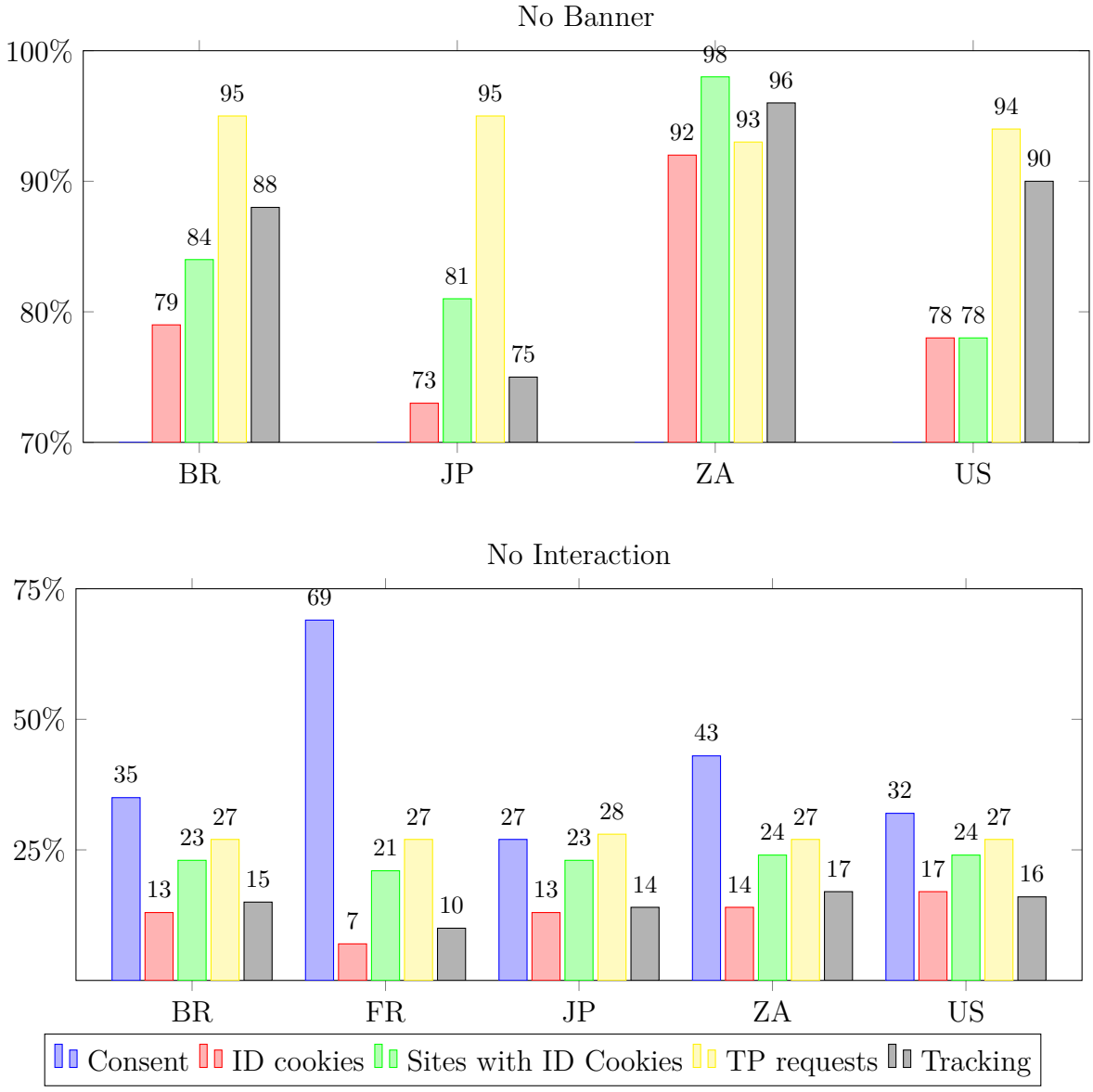


Figure 5.7: Prevalence of consent banners, ID cookies, sites serving ID cookies, third-party requests, and tracking requests in different scenarios: the upper figure shows the `No Banner` scenario, when no consent banner is shown in countries other than FR and consequently no consent prevalence is measured. The figure at the bottom shows the `No Interaction` scenario when a consent banner is shown but the user has not yet interacted with it.

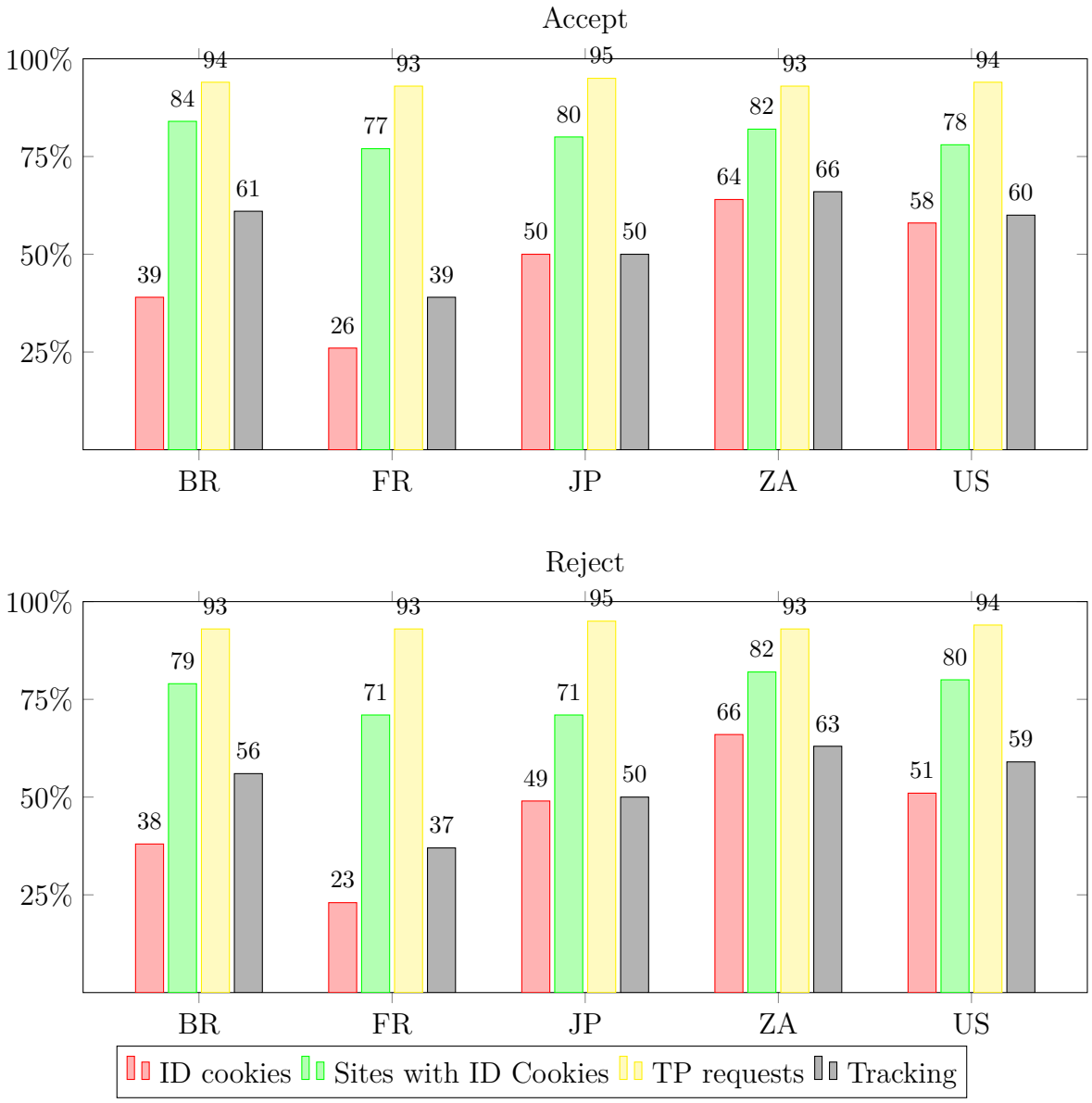


Figure 5.8: ID cookies, sites serving ID cookies, third-party requests, and tracking requests in different scenarios: the upper figure shows the `Accept` scenario while the figure at the bottom shows the `Reject` scenario.

results indicate that most third-party requests sent by the websites in the US are also tracking requests. This changes after interaction with the consent banner, where there is a surge of both third-party requests in the `accept` and `reject` cases. This indicates dense activities upon user consent and suggests that websites do not necessarily honor the user's choice of consent.

When there is no banner displayed, tracking mechanisms across countries seem to be uninhibited as seen in figure 5.9. This suggests that in the absence of regulations or enforcement, website operators rely heavily on third-party tools and tracking mechanisms. Similar to our previous discussion concerning the prevalence of the consent banners, the high number of HTTP requests including third-party and tracking in the US can be attributed to the advertising industry and the lack of a federal data protection regulation like GDPR in Europe. State-specific regulations such as California's CCPA do not specifically require explicit consent upfront. On the contrary, South Africa's POPI Act is similar to GDPR but its enforcement is not as stringent, hence the increase in HTTP requests and tracking.

5.5 Discussion

We explored the enforcement of cookie banners across continents, providing an overview of their adoption and implementation. One key observation from our study is the role of consent banners in notifying users about the use of cookies and other related disclaimers. Consent banners serve as a "garde fou" for website operators, essentially acting as a safety net. Instead of immediately initiating tracking, operators await user engagement. This works in regions with stringent regulations on data protection and user privacy, compelling website operators to secure user consent before conducting intrusive activities such as tracking. On the contrary, when users visit websites from areas without such regulations, website operators often do not always seek upfront consent.

Consequently, these websites revert to their older behavior patterns, relying heavily on third-party tools and tracking mechanisms. This differentiation in approach highlights disparity in user experience based on their geographic location and the regulatory environment. Furthermore, our study highlights economic incentives behind the adoption of certain tracking mechanisms. The reliance on third-party tools and tracking mechanisms, particularly in unregulated regions, might be driven by the potential gains from targeted advertising and data sharing. This economic angle warrants further exploration to understand the trade-offs between user privacy and revenue generation. In conclusion, while cookie banners have become a ubiquitous feature of the modern web, their enforcement and impact vary widely across regions. Striking a balance between regulatory compliance, user experience, and economic incentives remains a challenging task for website operators.

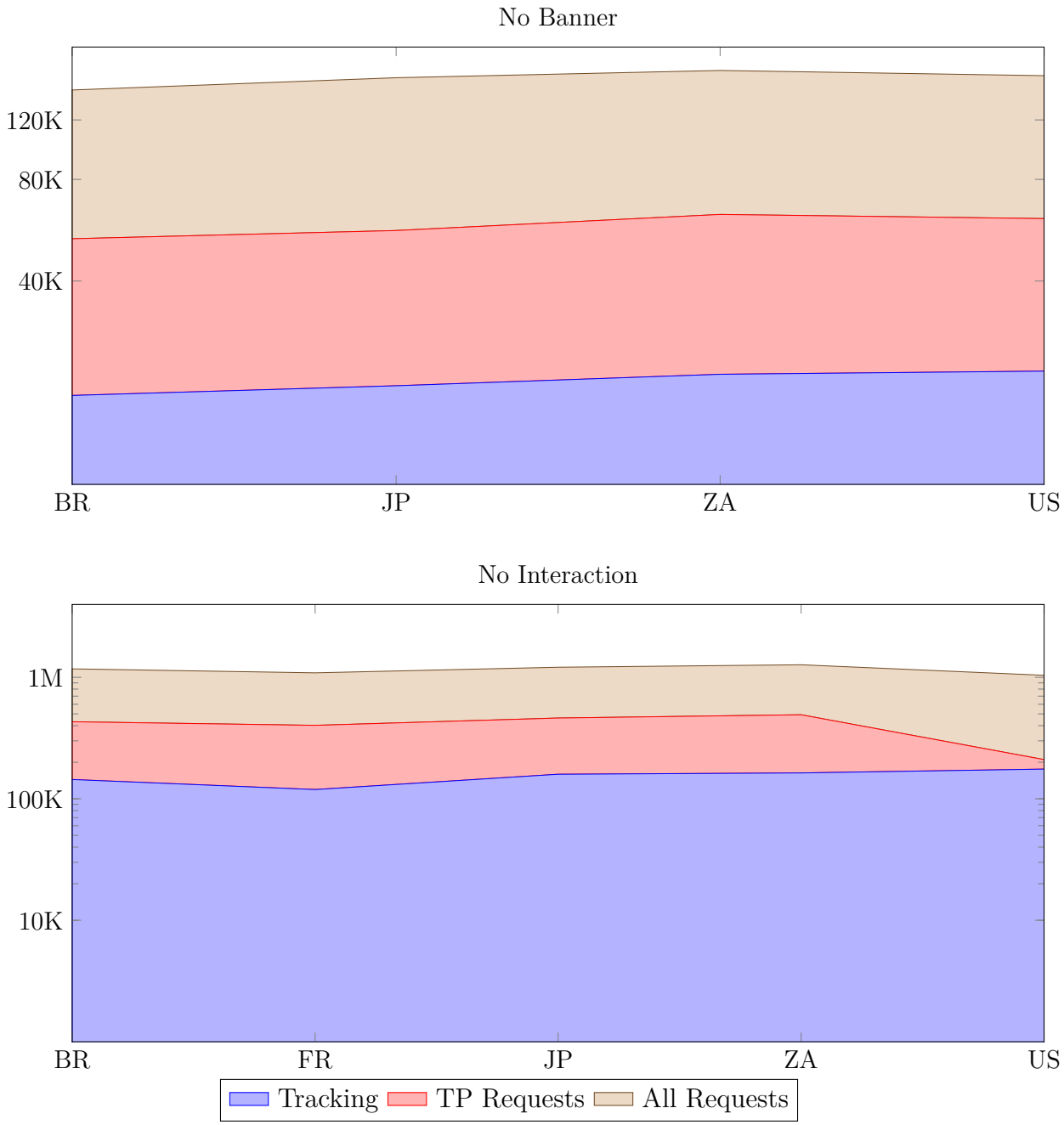


Figure 5.9: Comparative analysis of HTTP requests made during the crawl across five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US).

5.6 Threats to validity

A primary concern regarding the validity of our study is the reliance on the k-Means clustering technique that we used. While the method is robust, no clustering technique is error-free. The potential for misclassification, even if minimal, exists. Such misclassifications, however subtle, could introduce deviations in our data, thereby influencing the banner prevalence discussed in our study. Additionally, our reliance on the Google Translation API introduces another layer of potential complexity. Though this API offers

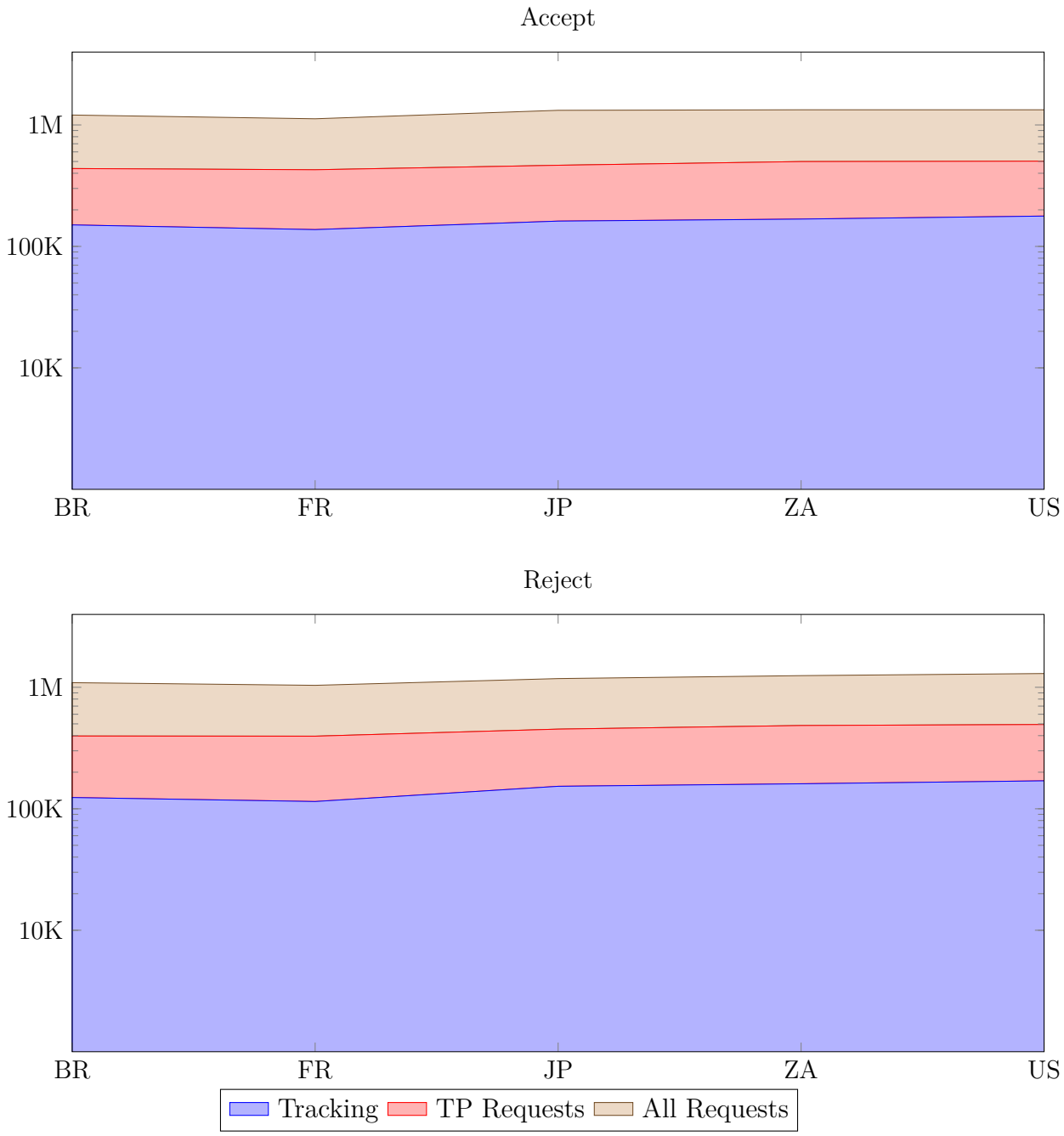


Figure 5.10: Comparative analysis of HTTP requests made during the crawl across five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US).

efficiency, it might not always capture the nuances of every language perfectly. This could lead to potential misinterpretations of cookie consent banners, especially when analyzing the wording and intent across diverse linguistic landscapes. Furthermore, the use of cookies and HTTP requests in our study can also be influenced by regional, cultural, or local intricacies in designing websites and creating content. Different regions might have unique design philosophies, content creation standards, or user engagement strategies, which could inadvertently affect the way cookies are utilized or HTTP requests are made.

5.7 Conclusion

Our study provides a comprehensive analysis of cookie consent banners on a global scale, emphasizing the impact of regional regulations on website behavior. Through automated visual detection across a vast dataset of web pages, we identified disparities in the enforcement and visibility of consent banners based on geographic location. The findings indicate that while many regions have adopted consent banners as a tool for transparency, their implementations vary. This variability can lead to diff user experiences and perceptions about online privacy. These disparities highlight the need for harmonized web privacy norms and more stringent, universally applicable regulations.

Chapter 6

Conclusion

The evolution of the web has been characterized by a dual trajectory: the advancement of technologies enhancing user experience and the increase in privacy risks introduced by these technologies. User tracking techniques have become more sophisticated. They leverage aspects of the user's browsing environment (UBE) such as device information, geolocation, and browser configurations. This raises concerns about the extent to which user information is exposed without explicit consent. Consequently, there is a growing demand for solutions that address these privacy risks while maintaining the seamless functionality that users expect.

In this thesis, we explored the impact of the UBE on user privacy by analyzing the interplay between usability and privacy. Our main objective was to determine the extent to which information exposure from the UBE is necessary for web page functionality, and how this exposure can be minimized to preserve user privacy. We contributed to a deeper understanding of how exposed information affects web page usability and to the broader discussion on protecting UBE access to enhance web privacy. One of our key findings is that not all UBE information is essential to web functionality. While certain attributes may be relevant, others provide little to no functional benefit and primarily serve tracking purposes. Determining this distinction is essential to limit unnecessary UBE exposure without compromising user experience.

Browser vendors are at the forefront of addressing these challenges. They face the dual responsibility of protecting user privacy and ensuring seamless website functionality. On one hand, they must implement measures to prevent tracking techniques that exploit UBE information. On the other hand, they need to maintain the flow of essential data that websites require to function properly. Striking this balance is complex, as overly restrictive mechanisms may cause website breakage, while lenient ones unnecessarily expose the UBE and threaten the user's privacy. Determining the relevance of UBE information each time it is about to be served is therefore essential. This involves assessing the necessity of requested information in real-time and deciding whether to allow or block it based on its impact on functionality. Such evaluation requires understanding the consequences of restricting each requested information. As we have seen in Chapter 3, implementing this

mechanism is fraught with challenges. UBE simulation and web functionality analysis are complex problems due to the vast diversity of UBE and the unpredictable ways in which websites may respond to changes in UBE. More work is needed to be done in simulating the UBE in such a way that it accurately reflects real user behavior. Additionally, analyzing website functionality to anticipate potential breakages demands predicting functional behavior by analyzing the JavaScript code on the website.

Given these complexities, user delegation can be an alternative short-term approach. By empowering users to make choices about the level of data exposure they are comfortable with, user privacy protections can be tailored to individual preferences. Users can decide how much functionality they are willing to compromise in exchange for enhanced privacy. For example, a user may choose to block certain UBE information even if it means occasional web page breakage. To facilitate informed decision-making, users would need transparent information about what data is collected and how it affects their browsing experience. Explaining the implications of blocking specific UBE information can help users understand the trade-offs involved. Our framework, discussed in Chapter 3 provides such transparency. This contributes to the creation of nuanced privacy controls that go beyond simple allow-or-block options.

6.1 Contributions

In this thesis, we aimed to explore the impact of the user's browsing environment on web privacy. Our primary objective was to investigate the relevance of the exposed information to website functionality. Our contributions can serve as groundwork for developing defense techniques against user privacy risks.

6.1.1 Determining information relevance in the user's browsing environment

We presented a novel approach to determine the relevance of UBE information concerning website functionality. Our aim was to learn how to enhance user privacy by minimizing information exposure. We categorized UBE information into geolocation, device, and browser attributes. To assess the impact of restricting specific UBE attributes, we simulated website visits using different UBE constructs by leveraging browser instrumentation tools like Playwright. This allowed us to systematically restrict specific UBE attributes and study their effects. Our methodology involved designing a web crawler that collects data from websites under these varying UBE constructs. We focused on static content by removing dynamic elements to ensure consistent analysis. We introduced Similarity Radar, a multidimensional tool that quantifies the impact of modifying UBE attributes across seven dimensions: HTML structure, JavaScript, CSS, visual rendering, cookies, HTTP requests, and textual content. By computing similarity scores for these dimensions,

we can determine how changes in UBE attributes affect website behavior. This helps restrict certain UBE attributes without negatively impacting website functionality. This approach provides a systematic framework to balance user privacy with website usability by identifying nonessential UBE information that can be safely restricted.

6.1.2 Exploring the impact of device information on the web

We explored the impact of the user agent (UA) string and associated information on websites to assess whether it remains essential for content adaptation and to evaluate its role in user tracking. Motivated by the contribution of UA information to browser fingerprinting, we aimed to determine if generalizing this information would affect website functionality. Our goal was to understand whether UA still serves its original purpose of tailoring content to specific browsers. To achieve this, we conducted a crawl of 270,048 web pages from 11,252 domains using three standard browsers — Chromium, Firefox, and WebKit — and their modified counterparts, referred to as "None-browsers", where UA and other identifying information were replaced with the word "None". We used the our novel framework described in the previous section and Similarity Radar to measure the relevance of UA. Our results revealed that before JavaScript execution, 100% of web pages were identical across all dimensions, indicating that servers no longer adapt content based on UA information. After JavaScript execution, we observed changes in 8.4% of web pages affecting visual rendering and HTML content. Further analysis showed that these changes were caused by third-party scripts related to ads, bot detection, and content delivery networks. We found that the usability issues were minimal and could be resolved by adopting browser-agnostic coding practices. Our study concludes that UA is no longer essential for website functionality, and retiring it could enhance user privacy without significantly impacting user experience.

6.1.3 Exploring the geolocation impact on enforcing privacy policies

We investigated the prevalence and enforcement of cookie consent banners across different continents and examined how the geolocation impacts the state of cookies and user tracking. Motivated by the inconsistencies in visibility of cookie banners in different geographic locations, and the limitations of previous studies that relied on manual observations or HTML/CSS inspection, we introduced a novel automated visual detection technique. Our methodology involved crawling 14,078 websites from five countries — Brazil, France, Japan, South Africa, and the United States — using OpenWPM. We used a combination of OpenCV for image processing and Tesseract OCR for text extraction to detect and classify cookie consent banners visually, and finally overcome language barriers through automatic translation. Our crawler ran in three distinct scenarios: without interacting with the cookie consent banner, accepting the banner, and rejecting it. Our results

revealed significant geographical disparities in the visibility of cookie consent banners and their enforcement. France exhibits the highest prevalence at 69%, due to strict GDPR regulations, while Japan has the lowest at 27%. We find that banner visibility correlates with the number of third-party cookies and tracking requests. The United States shows an increase in third-party cookies and tracking activities after user interaction with banners, an indication that websites do not honor user choices effectively. Our study underscores the importance of automated detection methods for accurate analysis in web privacy measurements. Additionally, there is a need for harmonized regulations to ensure consistent enforcement of user privacy protections worldwide.

6.2 Future work

In this thesis, we advanced our understanding of the interplay between UBE information and website behavior. However, our studies have also uncovered several areas that require further exploration to enhance privacy protections without compromising the user experience. The challenges range from improving the crawl validity, conducting more studies to draw out behavioral patterns of UBE information to website usability, to improving simulation of the UBE and analyzing web functionality. We discuss those points in detail in the following sections.

6.2.1 Short-term perspectives

In the short run, further research can build upon our findings to address current limitations and open questions. This could range from conducting studies on the absence of UA to examining cross-browser behavioral patterns to provide deeper insights into the impact of restricted UBE information in various browsers. Enhancing our methodologies for UBE simulation can also contribute to the accuracy of web privacy measurements in the short-term.

Removing UA entirely

A study should be conducted to explore the impact of having no UA information at all — i.e, setting the UA string and all identifying information to be empty, then null, then undefined — rather than generalizing it to a placeholder value like “None”. In Chapter 4, we investigated the impact of modifying UA and identifying information to “None” and observed that most websites continued to function correctly, suggesting that the significance of UA has diminished on the web. However, replacing the UA with a generic value still provides some information to websites, which may influence their behavior. The open question is, would the UA still be insignificant if its value exposed to the websites was empty, `null`, or `undefined`? Additionally, examining cross-browser behavioral patterns in this context could reveal more to the relevance of UA on the web.

Restricting the set I of irrelevant UBE attributes

In Chapter 3, we determined the relevance of individual UBE attributes by simulating their absence and observing changes in website behavior. This approach allowed us to identify the set I of irrelevant attributes that could be restricted without impacting functionality. A study should be conducted to evaluate the collective impact of restricting all attributes in the set I on website behavior. The aim would be to develop an effective defense against browser fingerprinting. Browser fingerprinting relies on collecting a wide array of UBE information to create unique identifiers for users [238], so such a study would answer the following research questions:

- Does restricting the set I of irrelevant UBE attributes lead to website breakage? If so, at what extent? What drives the breakages and why?
- What is the impact of this collective restriction on the uniqueness of browser fingerprints?

By restricting all irrelevant attributes, it could be observed whether cumulative effects lead to functionality issues not evident when attributes are restricted individually. Moreover, measuring the change in fingerprint uniqueness would provide more insights on the effectiveness of attribute restriction in enhancing web privacy. And lastly, such a study would provide evidence-based insights into the trade-offs needed to improve user privacy settings.

Better addressing bot detection

A short-term solution to bot detection could be to include randomness in our browsing patterns such as simulating realistic mouse movements, clicks, and interacting with web elements in a non-uniform manner. Additionally, we can simulate the UBE constructs using real browser fingerprints [87] where the crawler presents different, legitimate browser configurations to avoid detection. Collaborating with existing research on evading bot detection can help enhance the validity of our crawl process. Finally, integrating machine learning models that learn and replicate human interaction patterns on websites can further help improve our crawler's ability to pass bot detection tests.

Qualitative study on geolocation impact on privacy policy enforcement

It may be interesting to inquire the websites that intentionally change banner visibility, state of cookies and tracking depending on the user's geolocation. A study should be conducted by interviewing available representatives from those websites to understand if there are reasons for that.

6.2.2 Long-term perspectives

Advancements in artificial intelligence (AI) offer opportunities to enhance UBE simulation to make web measurements more accurate. Ongoing research can leverage AI to identify subtle cues that distinguish automated tools from real users. This would then help incorporate the findings in device emulation in such a way that crawlers can better avoid bot detection. But this is a cat and mouse game between web researchers and bot hunters. Bot hunters continuously develop sophisticated detection methods such as analyzing behavioral biometrics, detecting inconsistencies in UBE attributes, or employing machine learning models themselves to identify automated behavior [3, 237, 39]. More research is needed to explore the possibility of automating the UBE while maximizing the chance to pass most bot detection tests.

Functional analysis of web pages presents another challenge because predicting the functional behavior and interaction outcomes of HTML, CSS, and JavaScript — both individually and collectively — is inherently complex. This complexity is amplified when code is obfuscated or when multiple functionalities are bundled together [78]. Understanding the intended functionality of web page components is essential when evaluating the relevance of UBE attributes. As we have discussed in Section 3.6.2, without this understanding, we cannot claim fully determining the relevance of UBE attributes. Determining functional behavior would put closure to breakage concerns due to restricting irrelevant UBE attributes. That would lead to enhanced privacy controls and web privacy.

6.3 Concluding note

In conclusion, the future of the web depends on our ability to balance technologies with privacy protections. Understanding the interplay between the user's browsing environment and website functionality will continue to be relevant. As the race between trackers and protective measures continues, determining the distinction between relevant and irrelevant UBE attributes, hence tradeoffs between website functionality and privacy risks, is likely to inspire new designs of privacy controls.

Bibliography

[1] *A research-oriented top sites ranking hardened against manipulation*. URL: https://tranco-list.eu/ (visited on 09/2023).

[2] *A short history of the Web*. en. Jan. 2024. URL: https://www.home.cern/science/computing/birth-web/short-history-web (visited on 02/2024).

[3] Alejandro Acien, Aythami Morales, Julian Fierrez, and Ruben Vera-Rodriguez. "BeCAPTCHA-Mouse: Synthetic mouse trajectories and improved bot detection". In: *Pattern Recognition* 127 (2022), p. 108643.

[4] Sanchit Aggarwal et al. "Modern web-development using reactjs". In: *International Journal of Recent Research Aspects* 5.1 (2018), pp. 133–137.

[5] Abdul Haddi Amjad, Shaoor Munir, Zubair Shafiq, and Muhammad Ali Gulzar. "Blocking Tracking JavaScript at the Function Granularity". In: *arXiv preprint arXiv:2405.18385* (2024).

[6] Ryan Amos, Gunes Acar, Eli Lucherini, Mihir Kshirsagar, Arvind Narayanan, and Jonathan Mayer. "Privacy policies over time: Curation and analysis of a million-document dataset". In: *Proceedings of the Web Conference 2021*. 2021, pp. 2165–2176.

[7] Aaron Andersen. *History of the Browser User-Agent String*. URL: https://webaim.org/blog/user-agent-string-history/.

[8] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. "FPSelect: low-cost browser fingerprints for mitigating dictionary attacks against web authentication mechanisms". In: *Proceedings of the 36th Annual Computer Security Applications Conference*. 2020, pp. 627–642.

[9] Elbren Antonio, Arnel Fajardo, and Ruji Medina. "Tracking browser fingerprint using rule based algorithm". In: *2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE. 2020, pp. 225–229.

[10] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce 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*. 2020, pp. 680–695.

[11] Benjamin Baron and Mirco Musolesi. "Where you go matters: A study on the privacy implications of continuous location tracking". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.4 (2020), pp. 1–32.

[12] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. "Disguised chromium browser: Robust browser, flash and canvas fingerprinting protection". In: *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. 2016, pp. 37–46.

[13] Colin J Bennett. "Cookies, web bugs, webcams and cue cats: Patterns of surveillance on the world wide web". In: *Ethics and Information Technology* 3 (2001), pp. 195–208.

[14] Brent W Benson Jr. "Javascript". In: *ACM SIGPLAN Notices* 34.4 (1999), pp. 25–27.

[15] Carlos Bermejo Fernandez, Dimitris Chatzopoulos, Dimitrios Papadopoulos, and Pan Hui. "This Website Uses Nudging: MTurk Workers' Behaviour on Cookie Consent Notices". en. In: *Proceedings of the ACM on Human-Computer Interaction* 5.CSCW2 (Oct. 2021), pp. 1–22. DOI: 10.1145/3476087. URL: https://dl.acm.org/doi/10.1145/3476087 (visited on 06/11/2023).

[16] Tim Berners-Lee and Daniel W. Connolly. *RFC 1866 - Hypertext Markup Language - 2.0*. Request for Comments RFC 1866. Internet Engineering Task Force, Nov. 1995. DOI: 10.17487/RFC1866. URL: https://datatracker.ietf.org/doc/rfc1866.

[17] Benjamin Biegel, Quinten David Soetens, Willi Hornig, Stephan Diehl, and Serge Demeyer. "Comparison of similarity metrics for refactoring detection". In: *Proceedings of the 8th working conference on mining software repositories*. 2011, pp. 53–62.

[18] Philip Bille. "A survey on tree edit distance and related problems". In: *Theoretical computer science* 337.1-3 (2005), pp. 217–239.

[19] Dino Bollinger, Karel Kubicek, Carlos Cotrini, and David Basin. "Automating Cookie Consent and GDPR Violation Detection". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2893–2910. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/bollinger.

[20] *Bot detection test: verify if your bot is detected*. URL: https://deviceandbrowserinfo.com/are_you_a_bot (visited on 08/2024).

[21] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[22] *Brazilian General Data Protection Law (LGPD, English translation)*. URL: https://iapp.org/resources/article/brazilian-data-protection-law-lgpd-english-translation/ (visited on 02/2024).

[23] Sacha Brisset, Romain Rouvoy, Lionel Seinturier, and Renaud Pawlak. “SFTM: Fast matching of web pages using Similarity-based Flexible Tree Matching”. In: *Information Systems* 112 (2023), p. 102126.

[24] Justin Brookman, Phoebe Rouge, Aaron Alva, and Christina Yeung. “Cross-device tracking: Measurement and disclosures”. In: *Proceedings on Privacy Enhancing Technologies* (2017).

[25] David Buttler. *A short survey of document structure similarity algorithms*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2004.

[26] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. “Extracting content structure for web pages based on visual representation”. In: *Web Technologies and Applications: 5th Asia-Pacific Web Conference, APWeb 2003, Xian, China, April 23–25, 2003 Proceedings 5*. Springer. 2003, pp. 406–417.

[27] *California Consumer Privacy Act (CCPA)*. en. Oct. 2018. URL: https://oag.ca.gov/privacy/ccpa (visited on 05/2023).

[28] Edna Dias Canedo, Angelica Toffano Seidel Calazans, Ian Nery Bandeira, Pedro Henrique Teixeira Costa, and Eloisa Toffano Seidel Masson. “Guidelines adopted by agile teams in privacy requirements elicitation after the Brazilian general data protection law (LGPD) implementation”. In: *Requirements Engineering* 27.4 (2022), pp. 545–567.

[29] John Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[30] Marco Capece, Angelo Di Giovanni, Lorenzo Cirigliano, Luigi Napolitano, Roberto La Rocca, Massimiliano Creta, Gianluigi Califano, Felice Crocetto, Claudia Collà Ruvolo, Giuseppe Celentano, et al. “YouTube as a source of information on penile prosthesis”. In: *Andrologia* 54.1 (2022), e14246.

[31] Juan Miguel Carrascosa, Jakub Mikians, Ruben Cuevas, Vijay Erramilli, and Nikolaos Laoutaris. “I always feel like somebody’s watching me: measuring online behavioural advertising”. In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. 2015, pp. 1–13.

[32] Preeti S Chauhan and Nir Kshetri. “2021 state of the practice in data privacy and security”. In: *Computer* 54.8 (2021), pp. 125–132.

[33] *Check Single URL*. URL: https://sitelookup.mcafee.com/en/feedback/url?action=checksingle&sid=BF6DB84A3A60F9AEB8F69A93DA023455.

[34] Huan-Yuan Chen and Hong Yu. “Intent-based Web Page Summarization with Structure-Aware Chunking and Generative Language Models”. In: *Companion Proceedings of the ACM Web Conference 2023*. 2023, pp. 310–313.

[35] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. "Detecting filter list evasion with event-loop-turn granularity javascript signatures". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1715–1729.

[36] Yu Chen, Wei-Ying Ma, and Hong-Jiang Zhang. "Detecting web page structure for adaptive viewing on small form factor devices". In: *Proceedings of the 12th international conference on World Wide Web*. 2003, pp. 225–233.

[37] Shauvik Roy Choudhary, Husayn Versee, and Alessandro Orso. "WEBDIFF: Automated identification of cross-browser issues in web applications". In: *2010 IEEE International Conference on Software Maintenance*. IEEE. 2010, pp. 1–10.

[38] *Chrome device emulation*. en. URL: https://developer.chrome.com/docs/chromedriver/mobile-emulation (visited on 08/2024).

[39] Zi Chu, Steven Gianvecchio, and Haining Wang. "Bot or human? A behavior-based online bot detection system". In: *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday* (2018), pp. 432–449.

[40] *Cloud Translation*. en. URL: https://cloud.google.com/translate (visited on 05/2023).

[41] *Cloudflare Radar*. en. July 2023. URL: https://radar.cloudflare.com/ (visited on 09/2023).

[42] Mark Clow and Mark Clow. "Introducing Webpack". In: *Angular 5 Projects: Learn to Build Single Page Web Applications Using 70+ Projects* (2018), pp. 133–137.

[43] *Consent-O-Matic Extension*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/consent-o-matic/ (visited on 09/2023).

[44] *Cookie Quick Manager Extension*. URL: https://addons.mozilla.org/en-US/firefox/addon/cookie-quick-manager/ (visited on 09/15/2023).

[45] *CookieBlock Extension*. URL: https://addons.mozilla.org/en-US/firefox/addon/cookieblock/ (visited on 09/2023).

[46] *Cookies et traceurs : que dit la loi ?* fr. URL: https://www.cnil.fr/fr/cookies-et-autres-traceurs/regles/cookies/que-dit-la-loi (visited on 10/2023).

[47] Stephen W Crown. "Improving visualization skills of engineering graphics students using simple JavaScript web based games". In: *Journal of Engineering Education* 90.3 (2001), pp. 347–355.

[48] *CrUX Methodology*. en. June 2022. URL: https://developer.chrome.com/docs/crux/methodology/ (visited on 09/2023).

[49] *CSS Snapshot*. URL: https://www.w3.org/TR/CSS/.

[50] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry*. 2004, pp. 253–262.

[51] Amit Datta, Jianan Lu, and Michael Carl Tschantz. "Evaluating anti-fingerprinting privacy enhancing technologies". In: *The World Wide Web Conference*. 2019, pp. 351–362.

[52] Michelle De Bruyn et al. "The protection of personal information (POPI) act-impact on South Africa". In: *International Business & Economics Research Journal (IBER)* 13.6 (2014), pp. 1315–1340.

[53] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. "We value your privacy... now take some cookies: Measuring the GDPR's impact on web privacy". In: *arXiv preprint arXiv:1808.05096* (2018).

[54] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. "We value your privacy... now take some cookies: Measuring the GDPR's impact on web privacy". In: *arXiv preprint arXiv:1808.05096* (2018).

[55] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[56] Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli, and Roberto Scopigno. "SpiderGL: a JavaScript 3D graphics library for next-generation WWW". In: *Proceedings of the 15th International Conference on Web 3D Technology*. 2010, pp. 165–174.

[57] Yana Dimova, Gertjan Franken, Victor Le Pochat, Wouter Joosen, and Lieven Desmet. "Tracking the Evolution of Cookie-based Tracking on Facebook". In: *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. 2022, pp. 181–196.

[58] *DOM Standard*. URL: https://dom.spec.whatwg.org/.

[59] Georg Dotzler and Michael Philippsen. "Move-optimized source code tree differencing". In: *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*. 2016, pp. 660–671.

[60] Andrea Drmic, Marin Silic, Goran Delac, Klemo Vladimir, and Adrian S Kurdija. "Evaluating robustness of perceptual image hashing algorithms". In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2017, pp. 995–1000.

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

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

[63] *ECMAScript Language Specification*. URL: https://tc39.es/ecma262/.

[64] Magdalini Eirinaki and Michalis Vazirgiannis. "Web mining for web personalization". In: *ACM Transactions on Internet Technology (TOIT)* 3.1 (2003), pp. 1–27.

[65] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher II. "Leveraging user-session data to support web application testing". In: *IEEE Transactions on Software Engineering* 31.3 (2005), pp. 187–202.

[66] Steven Englehardt, Christian Eubank, Peter Zimmerman, Dillon Reisman, and Arvind Narayanan. "Web privacy measurement: Scientific principles, engineering platform, and new results". In: *Manuscript posted at http://randomwalker.info/publications/WebPrivacyMeasurement.pdf* 8 (2014), pp. 20–62.

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

[68] Steven Englehardt and Arvind Narayanan. "Online tracking: A 1-million-site measurement and analysis". In: *Proceedings of ACM CCS 2016*. 2016.

[69] *Enhanced Tracking Protection in Firefox for desktop | Firefox Help*. URL: https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop (visited on 11/2023).

[70] Steffie Jacob Eravuchira, Vaibhav Bajpai, Jürgen Schönwälder, and Sam Crawford. "Measuring web similarity from dual-stacked hosts". In: *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE. 2016, pp. 181–187.

[71] Abigayle Erickson. "Comparative Analysis of the EU's GDPR and Brazil's LGPD: Enforcement Challenges with the LGPD". In: *Brook. J. Int'l L.* 44 (2018), p. 859.

[72] Khyara F Passos. "Compliance with brazil's new data privacy legislation: What us companies need to know". In: *Compliance with Brazil's New Data Privacy Legislation: What US Companies Need to Know: F. Passos, Khyara*. [Sl]: SSRN, 2021.

[73] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. “Fine-grained and accurate source code differencing”. In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 2014, pp. 313–324.

[74] Heather Federman. “Examining Virginia’s New Data Protection Law”. In: *Risk Management* 68.6 (2021), pp. 8–9.

[75] Paul Ferguson and Geoff Huston. “What is a VPN?” In: (1998).

[76] Olga Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016.

[77] Imane Fouad, Cristiana Santos, Feras Al Kassar, Nataliia Bielova, and Stefano Calzavara. “On compliance of cookie purposes with the purpose specification principle”. In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2020, pp. 326–333.

[78] Romain Fouquet. “Improving Web User Privacy Through Content Blocking”. PhD thesis. Université de Lille, 2023.

[79] Romain Fouquet, Pierre Laperdrix, and Romain Rouvoy. “Breaking Bad: Quantifying the Addiction of Web Elements to JavaScript”. In: *ACM Transactions on Internet Technology* 23.1 (2023), pp. 1–28.

[80] Nathaniel Fruchter, Hsin Miao, Scott Stevenson, and Rebecca Balebako. “Variations in tracking in relation to geographic location”. In: *arXiv preprint arXiv:1506.04103* (2015).

[81] Kiran Garimella, Orestis Kostakis, and Michael Mathioudakis. “Ad-blocking: A study on performance, privacy and counter-measures”. In: *Proceedings of the 2017 ACM on Web Science Conference*. 2017, pp. 259–262.

[82] Jesse James Garrett et al. “Ajax: A new approach to web applications”. In: (2005).

[83] *GDPR Not Consent Extension*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/gdpr-not-consent/ (visited on 09/2023).

[84] *General Data Protection Regulation*. en. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj (visited on 05/2023).

[85] Phillipa Gill, Yashar Ganjali, and Bernard Wong. “Dude, Where’s That {IP}? Circumventing Measurement-based {IP} Geolocation”. In: *19th USENIX Security Symposium (USENIX Security 10)*. 2010.

[86] *Global Consent Manager*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/global-consent-manager/ (visited on 09/2023).

[87] *Global Statistics- Am I Unique ?* URL: https://amiunique.org/fingerprints-global-statistics (visited on 09/2024).

[88] Eric J Glover, Kostas Tsioutsiouliklis, Steve Lawrence, David M Pennock, and Gary W Flake. "Using web structure for classifying and describing web pages". In: *Proceedings of the 11th international conference on World Wide Web*. 2002, pp. 562–569.

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

[90] *Google Consent Dialog Remover Extension*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/google-consent-dialog-remover/ (visited on 09/2023).

[91] *google/diff-match-patch*. Jan. 2023. URL: https://github.com/google/diff-match-patch.

[92] Thamme Gowda and Chris A Mattmann. "Clustering web pages based on structure and style similarity (application paper)". In: *2016 IEEE 17th International conference on information reuse and integration (IRI)*. IEEE. 2016, pp. 175–180.

[93] Colin M. Gray, Cristiana Santos, Nataliia Bielova, Michael Toth, and Damian Clifford. "Dark Patterns and the Legal Requirements of Consent Banners: An Interaction Criticism Perspective". en. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Yokohama Japan: ACM, May 2021, pp. 1–18. ISBN: 978-1-4503-8096-6. DOI: 10.1145/3411764.3445779. URL: https://dl.acm.org/doi/10.1145/3411764.3445779 (visited on 07/2023).

[94] Brad Green and Shyam Seshadri. *AngularJS*. " O'Reilly Media, Inc.", 2013.

[95] Alan Grosskurth and Michael W Godfrey. "Architecture and evolution of the modern web browser". In: *Preprint submitted to Elsevier Science* 12.26 (2006), pp. 235–246.

[96] Saikat Guha, Bin Cheng, and Paul Francis. "Challenges in measuring online advertising systems". In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 2010, pp. 81–87.

[97] Hana Habib, Megan Li, Ellie Young, and Lorrie Cranor. ""Okay, whatever": An Evaluation of Cookie Consent Interfaces". en. In: *CHI Conference on Human Factors in Computing Systems*. New Orleans LA USA: ACM, Apr. 2022, pp. 1–27. ISBN: 978-1-4503-9157-3. DOI: 10.1145/3491102.3501985. URL: https://dl.acm.org/doi/10.1145/3491102.3501985 (visited on 08/2023).

[98] Alexander Hambley, Yeliz Yesilada, Markel Vigo, and Simon Harper. "Web Structure Derived Clustering for Optimised Web Accessibility Evaluation". In: *Proceedings of the ACM Web Conference*. 2023.

[99] Sakib Haque, Zachary Eberhart, Aakash Bansal, and Collin McMillan. "Semantic similarity metrics for evaluating source code summarization". In: *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 2022, pp. 36–47.

[100] Mahdi Hashemi. "Web page classification: a survey of perspectives, gaps, and future directions". In: *Multimedia Tools and Applications* 79.17-18 (2020), pp. 11921–11945.

[101] Waleed Hashmi, Moumena Chaqfeh, Lakshminarayanan Subramanian, and Yasir Zaki. "QLUE: A computer vision tool for uniform qualitative evaluation of web pages". In: *Proceedings of the ACM Web Conference 2022*. 2022, pp. 2400–2410.

[102] Philip Hausner and Michael Gertz. *Dark Patterns in the Interaction with Cookie Banners*. 2021. arXiv: 2103.14956 [cs.HC].

[103] Raymond Hill. *uBlock Origin (uBO)*. Aug. 2023. URL: https://github.com/gorhill/uBlock (visited on 07/2023).

[104] *Hoxx VPN*. URL: https://hoxx.com/ (visited on 09/2023).

[105] *HTTP Request Fields*. URL: https://www.w3.org/Protocols/HTTP/HTRQ_Headers.html.

[106] Boyang Hu, Qicheng Lin, Yao Zheng, Qiben Yan, Matthew Troglia, and Qingyang Wang. "Characterizing location-based mobile tracking in mobile ad networks". In: *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2019, pp. 223–231.

[107] *I Don't Care About Cookies Extension*. en-US. (Visited on 09/2023).

[108] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Yiannis Kompatsiaris. "Towards a framework for detecting advanced web bots". In: *Proceedings of the 14th international conference on availability, reliability and security*. 2019, pp. 1–10.

[109] *Improving user privacy and developer experience with User-Agent Client Hints*. en. June 2020. URL: https://developer.chrome.com/articles/user-agent-client-hints/ (visited on 07/2023).

[110] Jean Luc Intumwayase. *intumwa/cookie-prevalence*. en. URL: https://github.com/intumwa/cookie-prevalence (visited on 01/2024).

[111] Jean Luc Intumwayase. *intumwa/ua-radar*. Oct. 2022. URL: https://github.com/intumwa/ua-radar?tab=readme-ov-file#ua-radar-1 (visited on 01/2024).

[112] Jean Luc Intumwayase, Imane Fouad, Pierre Laperdrix, and Romain Rouvoy. "UA-Radar: Exploring the Impact of User Agents on the Web". In: *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*. 2023, pp. 31–43.

[113] Costas Iordanou, Georgios Smaragdakis, Ingmar Poese, and Nikolaos Laoutaris. "Tracing cross border web tracking". In: *Proceedings of the internet measurement conference 2018*. 2018, pp. 329–342.

[114] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. "A critical evaluation of website fingerprinting attacks". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 263–274.

[115] Jordan Jueckstock and Alexandros Kapravelos. "Visiblev8: In-browser monitoring of javascript in the wild". In: *Proceedings of the Internet Measurement Conference*. 2019, pp. 393–405.

[116] Georgios Kampanos and Siamak F. Shahandashti. "Accept All: The Landscape of Cookie Banners in Greece and the UK". In: *CoRR* abs/2104.05750 (2021). arXiv: 2104.05750. URL: https://arxiv.org/abs/2104.05750.

[117] Farzaneh Karegar, John Sören Pettersson, and Simone Fischer-Hübner. "The Dilemma of User Engagement in Privacy Notices: Effects of Interaction Modes and Habituation on User Attention". en. In: *ACM Transactions on Privacy and Security* 23.1 (Feb. 2020), pp. 1–38. DOI: 10.1145/3372296. URL: https://dl.acm.org/doi/10.1145/3372296 (visited on 06/2023).

[118] Ethan Katz-Bassett, John P John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. "Towards IP geolocation using delay and topology measurements". In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006, pp. 71–84.

[119] Rishabh Khandelwal, Asmit Nayak, Hamza Harkous, and Kassem Fawaz. "Automated Cookie Notice Analysis and Enforcement". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 1109–1126. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/khandelwal.

[120] Rishabh Khandelwal, Asmit Nayak, Hamza Harkous, and Kassem Fawaz. "Automated cookie notice analysis and enforcement". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 1109–1126.

[121] Rohit Khankhoje. "Web Page Element Identification Using Selenium and CNN: A Novel Approach". In: *Journal of Software* 1.1 (2023).

[122] Hyungsub Kim, Sangho Lee, and Jong Kim. "Exploring and mitigating privacy threats of HTML5 geolocation API". In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, pp. 306–315.

[123] Jung Il Kim and Eun Joo Lee. "An Approach to Detect Similar Script Functions in Web Applications Based on Calling Information". In: *Applied Mechanics and Materials* 263 (2013), pp. 1593–1599.

[124] Daniel Kladnik. *I don't care about cookies*. en. URL: https://www.i-dont-care-about-cookies.eu/ (visited on 06/2023).

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

[126] Konrad Kollnig, Anastasia Shuba, Max Van Kleek, Reuben Binns, and Nigel Shadbolt. "Goodbye tracking? Impact of iOS app tracking transparency and privacy labels". In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 2022, pp. 508–520.

[127] Yuriy Kotsarenko. "Measuring perceived color difference using YIQ NTSC transmission color space in mobile applications". In: *Programacion Matematica y Software* (2018).

[128] Ranjitha Kumar, Jerry O Talton, Salman Ahmad, and Scott R Klemmer. "Bricolage: example-based retargeting for web design". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2011, pp. 2197–2206.

[129] Ranjitha Kumar, Jerry O Talton, Salman Ahmad, Tim Roughgarden, and Scott R Klemmer. "Flexible tree matching". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence-Volume Volume Three*. 2011, pp. 2674–2679.

[130] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. "The quic transport protocol: Design and internet-scale deployment". In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196.

[131] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clémentine Maurice, Yossi Oren, Romain Rouvoy, Walter Rudametkin, and Yuval Yarom. "Drawnapart: A device identification technique based on remote gpu fingerprinting". In: *arXiv preprint arXiv:2201.09956* (2022).

[132] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. "Browser fingerprinting: A survey". In: *ACM Transactions on the Web (TWEB)* 14.2 (2020), pp. 1–33.

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

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

[135] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. "Fingerprinting in style: Detecting browser extensions via injected style sheets". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 2507–2524.

[136] Irvine Lapsley. "The NPM agenda: back to the future". In: *Financial accountability & management* 24.1 (2008), pp. 77–96.

[137] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. "Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web". In: *arXiv preprint arXiv:1811.00918* (2018).

[138] Guillaume Lavoué, Laurent Chevalier, and Florent Dupont. "Streaming compressed 3D data on the web using JavaScript and WebGL". In: *Proceedings of the 18th international conference on 3D web technology*. 2013, pp. 19–27.

[139] Marc T Law, Carlos Sureda Gutierrez, Nicolas Thome, Stéphane Gançarski, and Matthieu Cord. "Structural and visual similarity learning for web page archiving". In: *2012 10th International Workshop on Content-Based Multimedia Indexing (CBMI)*. IEEE. 2012, pp. 1–6.

[140] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS 2019. Feb. 2019. DOI: 10.14722/ndss.2019.23386.

[141] Sebastian Lekies, Ben Stock, and Martin Johns. "25 million flows later: large-scale detection of DOM-based XSS". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 1193–1204.

[142] Ada Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. "Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016". In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016.

[143] Xiao Li and Xiao Jing Zhong. "The source code plagiarism detection using AST". In: *2010 International symposium on intelligence information processing and trusted computing*. IEEE. 2010, pp. 406–408.

[144] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. "Good bot, bad bot: Characterizing automated browsing activity". In: *2021 IEEE symposium on security and privacy (sp)*. IEEE. 2021, pp. 1589–1605.

[145] Thomas Linden, Rishabh Khandelwal, Hamza Harkous, and Kassem Fawaz. "The privacy policy landscape after the GDPR". In: *arXiv preprint arXiv:1809.08396* (2018).

[146] Ling Liu, Calton Pu, and Wei Tang. "Webcq-detecting and delivering information changes on the web". In: *Proceedings of the ninth international conference on Information and knowledge management*. 2000, pp. 512–519.

[147] Qing Liu, Jing Wang, Dehai Zhang, Yun Yang, and NaiYao Wang. "Text features extraction based on TF-IDF associating semantic". In: *2018 IEEE 4th international conference on computer and communications (ICCC)*. IEEE. 2018, pp. 2338–2343.

[148] Ari Luotonen and Kevin Altis. "World-wide web proxies". In: *Computer Networks and ISDN systems* 27.2 (1994), pp. 147–154.

[149] Finlay Macklon, Mohammad Reza Taesiri, Markos Viggiato, Stefan Antoszko, Natalia Romanova, Dale Paas, and Cor-Paul Bezemer. "Automatically Detecting Visual Bugs in HTML5 Canvas Games". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–11.

[150] Aniss Maghsoudlou, Lukas Vermeulen, Ingmar Poese, and Oliver Gasser. *Characterizing the VPN Ecosystem in the Wild*. Feb. 2023. URL: http://arxiv.org/abs/2302.06566 (visited on 08/2023).

[151] Masood Mansoori and Ian Welch. "How do they find us? A study of geolocation tracking techniques of malicious web sites". In: *Computers & Security* 97 (2020), p. 101948.

[152] Florencia Marotta-Wurgler. "Self-regulation and competition in privacy policies". In: *The Journal of Legal Studies* 45.S2 (2016), S13–S39.

[153] Célestin Matte, Nataliia Bielova, and Cristiana Santos. "Do Cookie Banners Respect my Choice? Measuring Legal Compliance of Banners from IAB Europe's Transparency and Consent Framework". In: *arXiv preprint arXiv:1911.09964* (2019).

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

[155] Paul W McBurney and Collin McMillan. "An empirical study of the textual similarity between source code and source code summaries". In: *Empirical Software Engineering* 21 (2016), pp. 17–42.

[156] Nicolas Merz, Zach Yale, Heather Geiger, Darrien Park, Kurt Blair, and Daniel Kailly. "Chrome's Proposed Feature-by uBlock Origin". In: (2018).

[157] *Minimal Consent Extension*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/minimal-consent/ (visited on 09/2023).

[158] James A Muir and Paul C Van Oorschot. "Internet geolocation: Evasion and counterevasion". In: *Acm computing surveys (csur)* 42.1 (2009), pp. 1–23.

[159] Shaoor Munir, Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. "CookieGraph: Understanding and Detecting First-Party Tracking Cookies". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2023, pp. 3490–3504.

[160] Eugene W Myers. "An O (ND) difference algorithm and its variations". In: *Algorithmica* 1.1-4 (1986), pp. 251–266.

[161] Paarth Naithani. "Curtailing the cookie monster through data protection by default". In: *Tilburg Law Review* 27.1 (2022), pp. 22–36.

[162] Michael Nebeling and Moira C Norrie. "Responsive design and development: methods, technologies and current issues". In: *Web Engineering: 13th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings 13*. Springer. 2013, pp. 510–513.

[163] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. *RFC 1945 - Hypertext Transfer Protocol –HTTP/1.0*. Request for Comments RFC 1945. Internet Engineering Task Force, May 1996. DOI: 10.17487/RFC1945. URL: https://datatracker.ietf.org/doc/rfc1945.

[164] Nick NIKIFORAKIS, A Kapravelos, W Joosen, C Kruegel, F Piessens, and G Vigna. *Cookieless monster: Exploring the ecosystem of web-based device fingerprinting 2013 IEEE Symposium on Security and Privacy*. 2013.

[165] Alexandra Nisenoff, Arthur Borem, Madison Pickering, Grant Nakanishi, Maya Thumpasery, and Blase Ur. "Defining "Broken": User Experiences and Remediation Tactics When Ad-Blocking or Tracking-Protection Tools Break a Website's User Experience". In: *Proceedings of the 32nd USENIX Security Symposium*. 2023.

[166] *NordVPN Official Website*. en. URL: https://nordvpn.com/ (visited on 04/2023).

[167] Midas Nouwens, Rolf Bagge, Janus Bager Kristensen, and Clemens Nylandsted Klokmose. "Consent-o-matic: Automatically answering consent pop-ups using adversarial interoperability". In: *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 2022, pp. 1–7.

[168] Sean O'Connor, Ryan Nurwono, Aden Siebel, and Eleanor Birrell. "(Un) clear and (In) conspicuous: The right to opt-out of sale under CCPA". In: *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*. 2021, pp. 59–72.

[169] Ognjen Pantelic, Kristina Jovic, and Stefan Krstovic. "Cookies implementation analysis and the impact on user privacy regarding GDPR and CCPA regulations". In: *Sustainability* 14.9 (2022), p. 5015.

[170] Emmanouil Papadogiannakis, Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P Markatos. "User tracking in the post-cookie era: How websites bypass gdpr consent to track users". In: *Proceedings of the web conference 2021*. 2021, pp. 2130–2141.

[171] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos Markatos. “Cookie synchronization: Everything you always wanted to know but were afraid to ask”. In: *The World Wide Web Conference*. 2019, pp. 1432–1442.

[172] Seongsoo Park, Seungcheol Ko, Jungsik Choi, Hwansoo Han, Seong-Je Cho, and Jongmoo Choi. “Detecting source code similarity using code abstraction”. In: *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*. 2013, pp. 1–9.

[173] *Playwright device emulation*. en. URL: https://github.com/microsoft/playwright/blob/main/packages/playwright-core/src/server/deviceDescriptorsSource.json (visited on 08/2024).

[174] *Playwright Library*. URL: https://playwright.dev/docs/api/class-playwright.

[175] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. “Tranco: A research-oriented top sites ranking hardened against manipulation”. In: *arXiv preprint arXiv:1806.01156* (2018).

[176] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. “IP geolocation databases: Unreliable?” In: *ACM SIGCOMM Computer Communication Review* 41.2 (2011), pp. 53–56.

[177] *Polish Cookie Consent Extension*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/polish-cookie-consent/ (visited on 09/2023).

[178] *Proton VPN*. en. URL: https://protonvpn.com (visited on 09/2023).

[179] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. “Long-term observation on browser fingerprinting: Users’ trackability and perspective”. In: *Proceedings on Privacy Enhancing Technologies* (2020).

[180] *Puppeteer*. en. URL: https://pptr.dev/ (visited on 08/2024).

[181] *Puppeteer device emulation*. en. URL: https://pptr.dev/api/puppeteer.knowndevices (visited on 08/2024).

[182] Chenxiong Qian, Hyungjoon Koo, ChangSeok Oh, Taesoo Kim, and Wenke Lee. “Slimium: debloating the chromium browser with feature subsetting”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 461–476.

[183] Rohit Rai. *Socket. IO real-time web application development*. Packt Publishing Ltd, 2013.

[184] Juan Ramos et al. “Using tf-idf to determine word relevance in document queries”. In: *Proceedings of the first instructional conference on machine learning*. Vol. 242. 1. Citeseer. 2003, pp. 29–48.

[185] A Rasaii. *BannerClick Extension*. URL: https://github.com/bannerclick/bannerclick (visited on 09/2023).

[186] Ali Rasaii, Shivani Singh, Devashish Gosain, and Oliver Gasser. *Exploring the Cookieverse: A Multi-Perspective Analysis of Web Cookies*. arXiv:2302.05353 [cs]. Feb. 2023. URL: http://arxiv.org/abs/2302.05353 (visited on 08/2023).

[187] Davi De Castro Reis, Paulo Braz Golgher, Altigran Soares Silva, and AlbertoF Laender. "Automatic web news extraction using tree edit distance". In: *Proceedings of the 13th international conference on World Wide Web*. 2004, pp. 502–511.

[188] *requests: Python HTTP for Humans*. URL: https://requests.readthedocs.io (visited on 08/2024).

[189] *RFC 791 - Internet Protocol*. Request for Comments RFC 791. Internet Engineering Task Force. DOI: 10.17487/RFC0791. URL: https://datatracker.ietf.org/doc/rfc791.

[190] *RFC 9293 - Transmission Control Protocol*. Request for Comments RFC 9293. Internet Engineering Task Force. DOI: 10.17487/RFC9293. URL: https://datatracker.ietf.org/doc/rfc9293.

[191] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: How to plan, design, and conduct effective tests*. John Wiley & Sons, 2008.

[192] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval". In: *International journal of computer vision* 40 (2000), pp. 99–121.

[193] Jukka Ruohonen and Kalle Hjerppe. *Predicting the Amount of GDPR Fines*. 2020. arXiv: 2003.05151 [cs.CY].

[194] Kimberly Ruth, Deepak Kumar, Brandon Wang, Luke Valenta, and Zakir Durumeric. "Toppling top lists: Evaluating the accuracy of popular website lists". In: *Proceedings of the 22nd ACM Internet Measurement Conference*. 2022, pp. 374–387.

[195] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. "A First Look at QUIC in the Wild". In: *Passive and Active Measurement: 19th International Conference, PAM 2018, Berlin, Germany, March 26–27, 2018, Proceedings 19*. Springer. 2018, pp. 255–268.

[196] Iskander Sanchez-Rola, Matteo Dell'Amico, Platon Kotzias, Davide Balzarotti, Leyla Bilge, Pierre-Antoine Vervier, and Igor Santos. "Can I opt out yet? GDPR and the global illusion of cookie control". In: *Proceedings of the 2019 ACM Asia conference on computer and communications security*. 2019, pp. 340–351.

[197] Cristiana Santos, Arianna Rossi, Lorena Sanchez Chamorro, Kerstin Bongard-Blanchy, and Ruba Abu-Salma. "Cookie banners, what's the purpose? analyzing cookie banner text through a legal lens". In: *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*. 2021, pp. 187–194.

[198] Cristiana Santos, Arianna Rossi, Lorena Sanchez Chamorro, Kerstin Bongard-Blanchy, and Ruba Abu-Salma. "Cookie banners, what's the purpose? analyzing cookie banner text through a legal lens". In: *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*. 2021, pp. 187–194.

[199] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D. Strowes, and Narseo Vallina-Rodriguez. "A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists". en. In: *Proceedings of the Internet Measurement Conference 2018*. Boston MA USA: ACM, Oct. 2018, pp. 478–493. ISBN: 978-1-4503-5619-0.

[200] Asuman Senol and Gunes Acar. "Unveiling the Impact of User-Agent Reduction and Client Hints: A Measurement Study". In: *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*. 2023, pp. 91–106.

[201] Kyle Simpson. *You Don't Know JS: ES6 & Beyond*. " O'Reilly Media, Inc.", 2015.

[202] Kristina P Sinaga and Miin-Shen Yang. "Unsupervised K-means clustering algorithm". In: *IEEE access* 8 (2020), pp. 80716–80727.

[203] Than Htut Soe, Oda Elise Nordberg, Frode Guribye, and Marija Slavkovik. "Circumvention by design - dark patterns in cookie consent for online news outlets". en. In: *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*. Tallinn Estonia: ACM, Oct. 2020, pp. 1–12. ISBN: 978-1-4503-7579-5. DOI: 10.1145/3419249.3420132. URL: https://dl.acm.org/doi/10.1145/3419249.3420132 (visited on 08/2023).

[204] Than Htut Soe, Cristiana Teixeira Santos, and Marija Slavkovik. *Automated detection of dark patterns in cookie banners: how to do it poorly and why it is hard to do it any other way*. 2022. arXiv: 2204.11836 [cs.LG].

[205] Yuri Son, Geumhwan Cho, Hyoungshick Kim, and Simon Woo. "Understanding Users' Risk Perceptions about Personal Health Records Shared on Social Networking Services". In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 2019, pp. 352–365.

[206] Jingkuan Song, Yi Yang, Zi Huang, Heng Tao Shen, and Richang Hong. "Multiple feature hashing for real-time large scale near-duplicate video retrieval". In: *Proceedings of the 19th ACM international conference on Multimedia*. 2011, pp. 423–432.

[207] Jannick Sørensen and Sokol Kosta. "Before and after gdpr: The changes in third party presence at public and private european websites". In: *The World Wide Web Conference*. 2019, pp. 1590–1600.

[208] Jannick Kirk Sørensen and Hilde Van den Bulck. "Public service media online, advertising and the third-party user data business: A trade versus trust dilemma?" In: *Convergence* 26.2 (2020), pp. 421–447.

[209] Mukund Srinath, Shomir Wilson, and C Lee Giles. "Privacy at scale: Introducing the privaseer corpus of web privacy policies". In: *arXiv preprint arXiv:2004.11131* (2020).

[210] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. "Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat". In: *The World Wide Web Conference*. 2019, pp. 3244–3250.

[211] Thomas Steiner, Anssi Kostiainen, and Marijn Kruisselbrink. "Geolocation in the Browser". In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, pp. 913–918.

[212] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. "Impact of similarity measures on web-page clustering". In: *Workshop on artificial intelligence for web search (AAAI 2000)*. Vol. 58. 2000, p. 64.

[213] Keen Sung, Joydeep Biswas, Erik Learned-Miller, Brian N Levine, and Marc Liberatore. "Server-side traffic analysis reveals mobile location information over the internet". In: *IEEE Transactions on Mobile Computing* 18.6 (2018), pp. 1407–1418.

[214] *Super Agent Extension*. en-US. URL: https://addons.mozilla.org/en-US/firefox/addon/super-agent/ (visited on 09/2023).

[215] Satoshi Suzuki et al. "Topological structural analysis of digitized binary images by border following". In: *Computer vision, graphics, and image processing* 30.1 (1985), pp. 32–46.

[216] Satoshi Suzuki et al. "Topological structural analysis of digitized binary images by border following". In: *Computer vision, graphics, and image processing* 30.1 (1985), pp. 32–46.

[217] Kuo-Chung Tai. "The tree-to-tree correction problem". In: *Journal of the ACM (JACM)* 26.3 (1979), pp. 422–433.

[218] *Tesseract documentation*. en-US. URL: https://tesseract-ocr.github.io/ (visited on 05/2023).

[219] Anastasios Tombros and Zeeshan Ali. "Factors affecting web page similarity". In: *Advances in Information Retrieval: 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005. Proceedings 27*. Springer. 2005, pp. 487–501.

[220] Ryan Torok and Amit Levy. "Only Pay for What You Leak: Leveraging Sandboxes for a Minimally Invasive Browser Fingerprinting Defense". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 1023–1040.

[221] *Tracking Prevention Policy*. Aug. 2019. URL: https://webkit.org/tracking-prevention-policy/ (visited on 11/2023).

[222] Willemijn Tutuarima. “Measuring Accessibility of Popular Websites when using ProtonVPN”. In: (2021).

[223] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. “Measuring the impact of the gdpr on data sharing in ad networks”. In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 2020, pp. 222–235.

[224] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. “The unwanted sharing economy: An analysis of cookie syncing and user transparency under GDPR”. In: *arXiv preprint arXiv:1811.08660* (2018).

[225] *Usage Statistics of JavaScript as Client-side Programming Language on Websites, February 2024*. URL: https://w3techs.com/technologies/details/cp-javascript (visited on 02/2024).

[226] *User-Agent Reduction*. URL: https://www.chromium.org/updates/ua-reduction/ (visited on 11/2023).

[227] Christine Utz, Martin Degeling, Sascha Fahl, Florian Schaub, and Thorsten Holz. “(Un) informed consent: Studying GDPR consent notices in the field”. In: *Proceedings of the 2019 acm sigsac conference on computer and communications security*. 2019, pp. 973–990.

[228] Christine Utz, Martin Degeling, Sascha Fahl, Florian Schaub, and Thorsten Holz. “(Un) informed consent: Studying GDPR consent notices in the field”. In: *Proceedings of the 2019 acm sigsac conference on computer and communications security*. 2019, pp. 973–990.

[229] Cyril Vallez, Andrei Kucharavy, and Ljiljana Dolamic. “Needle In A Haystack, Fast: Benchmarking Image Perceptual Similarity Metrics At Scale”. In: *arXiv preprint arXiv:2206.00282* (2022).

[230] Rob Van Eijk, Hadi Asghari, Philipp Winter, and Arvind Narayanan. “The impact of user location on cookie notices (inside and outside of the European Union)”. In: *arXiv preprint arXiv:2110.09832* (2021).

[231] Naushad Varish and Arup Kumar Pal. “A content based image retrieval using color and texture features”. In: *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*. 2016, pp. 1–7.

[232] Matteo Varvello, Iñigo Querejeta Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Goncalo Pestana, and Benjamin Livshits. “VPN-Zero: A Privacy-Preserving Decentralized Virtual Private Network”. In: *2021 IFIP Networking Conference (IFIP Networking)*. IEEE. 2021, pp. 1–6.

[233] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. “{FP-Scanner}: The privacy implications of browser fingerprint inconsistencies”. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 135–150.

[234] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. "{FP-Scanner}: The privacy implications of browser fingerprint inconsistencies". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 135–150.

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

[236] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. "FP-Crawlers: studying the resilience of browser fingerprinting to block crawlers". In: *MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web*. 2020.

[237] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. "Fp-crawlers: studying the resilience of browser fingerprinting to block crawlers". In: *MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web*. 2020.

[238] *Visualize your browser and device fingerprint*. URL: https://deviceandbrowserinfo.com/info_device (visited on 09/2024).

[239] W Gregory Voss. "First the GDPR, now the proposed ePrivacy regulation". In: *Journal of Internet Law* 21.1 (2017), pp. 3–11.

[240] Asma AI Vranaki and Francesca Farmer. "The Decline of Third-Party Cookies in the AdTech Sector: Of Data Protection Law and Regulation (I)". In: *Available at SSRN 4414566* (2022).

[241] Vytautas Vyšniauskas. "Anti-aliased pixel and intensity slope detector". In: *Elektronika ir elektrotechnika* 95.7 (2009), pp. 107–110.

[242] Tim Wambach and Katharina Bräunlich. "The evolution of third-party web tracking". In: *Information Systems Security and Privacy: Second International Conference, ICISSP 2016, Rome, Italy, February 19-21, 2016, Revised Selected Papers 2*. Springer. 2017, pp. 130–147.

[243] Haoyu Wang, Mengxin Liu, Yao Guo, and Xiangqun Chen. "Similarity-based web browser optimization". In: *Proceedings of the 23rd international conference on World wide web*. 2014, pp. 575–584.

[244] Xinfei Wang. "A survey of online advertising click-through rate prediction models". In: *2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*. Vol. 1. IEEE. 2020, pp. 516–521.

[245] Yong Wang, Daniel Burgener, Marcel Flores, Aleksandar Kuzmanovic, and Cheng Huang. "Towards {Street-Level}{Client-Independent}{IP} Geolocation". In: *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. 2011.

[246] Zilun Wang, Wei Meng, and Michael R Lyu. "Fine-Grained Data-Centric Content Protection Policy for Web Applications". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2023, pp. 2845–2859.

[247] Steve Ward and Mat Hostetter. "Curl: a language for web content". In: *International journal of Web engineering and technology* 1.1 (2003), pp. 41–62.

[248] Michael Weeks. "Creating a web-based, 2-D action game in JavaScript with HTML5". In: *Proceedings of the 2014 ACM Southeast Regional Conference*. 2014, pp. 1–6.

[249] Wu Wen, Xiaobo Xue, Ya Li, Peng Gu, and Jianfeng Xu. "Code similarity detection using ast and textual information". In: *International Journal of Performability Engineering* 15.10 (2019), p. 2683.

[250] Liu Wenyin, Guanglin Huang, Liu Xiaoyue, Zhang Min, and Xiaotie Deng. "Detection of phishing webpages based on visual similarity". In: *Special interest tracks and posters of the 14th international conference on World Wide Web*. 2005, pp. 1060–1061.

[251] Vera Wesselkamp, Imane Fouad, Cristiana Santos, Yanis Boussad, Nataliia Bielova, and Arnaud Legout. "In-depth technical and legal analysis of tracking on health related websites with ernie extension". In: *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*. 2021, pp. 151–166.

[252] *Wget - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/software/wget/ (visited on 08/2024).

[253] Tingmin Wu, Rongjunchen Zhang, Wanlun Ma, Sheng Wen, Xin Xia, Cecile Paris, Surya Nepal, and Yang Xiang. "What risk? i don't understand. an empirical study on users' understanding of the terms used in security texts". In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 2020, pp. 248–262.

[254] Feng Xiao, Jianwei Huang, Yichang Xiong, Guangliang Yang, Hong Hu, Guofei Gu, and Wenke Lee. "Abusing hidden properties to attack the node. js ecosystem". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 2951–2968.

[255] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. "OpenVPN is Open to VPN Fingerprinting". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 483–500. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/xue-diwen.

[256] Tetsuo Yamamoto, Makoto Matsushita, Toshihiro Kamiya, and Katsuro Inoue. "Measuring similarity of large software systems based on source code correspondence". In: *Product Focused Software Process Improvement: 6th International Conference, PROFES 2005, Oulu, Finland, June 13-15, 2005. Proceedings 6*. Springer. 2005, pp. 530–544.

[257] Li Yujian and Liu Bo. "A normalized Levenshtein distance metric". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1091–1095.

[258] Muhammad Zain ul Abideen, Shahzad Saleem, and Madiha Ejaz. "Vpn traffic detection in ssl-protected channel". In: *Security and Communication Networks* 2019 (2019), pp. 1–17.

[259] Yujing Zeng, Jie Gao, and Chunsong Wu. "Responsive web design and its use by an e-commerce website". In: *Cross-Cultural Design: 6th International Conference, CCD 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings 6*. Springer. 2014, pp. 509–519.

[260] Frederik J Zuiderveen Borgesius, Sanne Kruikemeier, Sophie C Boerman, and Natali Helberger. "Tracking walls, take-it-or-leave-it choices, the GDPR, and the ePrivacy regulation". In: *Eur. Data Prot. L. Rev.* 3 (2017), p. 353.

Appendix A

Appendices

A Exploring the impact of device information on the web

Code repository for UA-Radar can be found on GitHub[1]. The artifacts include various forms of data and code that were used in the experiments, as well as detailed documentation and instructions to help others reproduce and validate the results.

A.1 Crawled Domains

A total of 12,000 domains were crawled to gather data and examine the relevance of the UA and associated information on the web. The websites selected for crawling were randomly chosen from the *Tranco*[2] list in April 2022. A comprehensive list of the crawled domains can be found in the file `./crawled_domains.json`. We used *McAfee SmartFilter*[3] to obtain the Internet categories of the crawled domains. Table A.1 shows the list of the categories of the crawled domains.

A.2 Navigator Properties Exposed During the Crawl

To examine the relevance of the user agent on the web, we instrumented the so-called "None-browsers"—a browser with just the string "None" in place of the user agent and any other browser identifying information. To instrument the None browsers, we modified the User-Agent HTTP request-header field 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

[1] https://github.com/intumwa/ua-radar
[2] https://tranco-list.eu/
[3] https://sitelookup.mcafee.com/en/feedback/url?action=checksingle&sid=BF6DB84A3A60F9AEB8F69A93DA023455

Table A.1: Categories of Crawled Domains

CATEGORY	DOMAINS
Internet Services	1,575
Business	1,419
Marketing	1,230
Education	724
News	702
Entertainment	631
Blogs	583
Games	491
Ads	459
Online Shopping	443
Information	434
Social Networking	355
Search Engines	352
Health	304
Miscellaneous	259
Malicious Sites	250
Finance	234
Pornography	207
Online Forum	204
Government	201
Non-Profit	166
No Category	35
TOTAL	11,252

word "None" on each of those properties. Furthermore, to avoid our modified browsers from being detected as bots, we set `navigator.webdriver` to `false`. Tables A.2, A.3, and A.4 detail the navigator properties exposed on all browsers during the crawl.

A.3 UA-Radar

UA-Radar is a Node.js application that relies on a Node.js add-on written in C++. The add-on takes care of the comparison tasks for each dimension of UA-Radar (Visual similarity, HTML structure, JavaScript, and CSS). To test the comparisons, we use Docker[35]. Below are instructions to produce results.

A.4 Running the Application

Move into the directory `./similarity`, build a Docker image[36], and run it.

```
cd similarity
docker build -t similarity .
docker run similarity
```

[35] https://www.docker.com/get-started/
[36] https://hub.docker.com/r/uaradar/similarity

Table A.2: Navigator Properties for Chromium Browsers

Navigator Property	Chromium	Chromium-None
appCodeName	Mozilla	Mozilla
appName	Netscape	Netscape
appVersion	5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/101.0.4951.15 Safari/537.36[4]	None
buildID	-	-
connection	NetworkInformation {onchange: null, effectiveType: '4g', rtt: 100, downlink: 1.5, saveData: false}	NetworkInformation {onchange: null, effectiveType: '4g', rtt: 100, downlink: 1.5, saveData: false}
cookieEnabled	TRUE	TRUE
doNotTrack	null	null
geolocation	Geolocation {}	Geolocation {}
hardwareConcurrency	8	8
ink	Ink {}	Ink {}
language	en-US	en-US
languages	['en-US']	['en-US']
maxTouchPoints	0	0
mediaCapabilities	MediaCapabilities {}	MediaCapabilities {}
mediaSession	MediaSession {metadata: null, playbackState: 'none'}	MediaSession {metadata: null, playbackState: 'none'}
mimeTypes	MimeTypeArray {0: MimeType, 1: MimeType, application/pdf: MimeType, text/pdf: MimeType, length: 2}	MimeTypeArray {0: MimeType, 1: MimeType, application/pdf: MimeType, text/pdf: MimeType, length: 2}
onLine	TRUE	TRUE
pdfViewerEnabled	TRUE	TRUE
oscpu	-	-
permissions	Permissions {}	Permissions {}
platform	Linux x86_64[5]	None[6]
plugins	Plugins {}	Plugins {}
product	Gecko	Gecko
productSub	20030107	20030107
scheduling	Scheduling {}	Scheduling {}
userAgent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/101.0.4951.15 Safari/537.36	None[7]
vendor	Google Inc.[8]	None[9]
vendorSub	-	-
webdriver	TRUE	TRUE

A.5 Changing Test Files

In the file `./similarity/app.js`, the visual comparison is the default UA-Radar dimension available as follows:

```
const compare = require('./index.js');


(async () => {
```

Table A.3: Navigator Properties for Firefox Browsers

Navigator Property	Firefox	Firefox-None
appCodeName	Mozilla	Mozilla
appName	Netscape	Netscape
appVersion	5.0 (X11)[10]	None
buildID	20181001000000	20181001000000
connection	-	-
cookieEnabled	TRUE	TRUE
doNotTrack	unspecified[11]	unspecified[12]
geolocation	Geolocation {}	Geolocation {}
hardwareConcurrency	8	8
ink	-	-
language	en-US	en-US
languages	['en-US', 'en']	['en-US', 'en']
maxTouchPoints	0	0
mediaCapabilities	MediaCapabilities {}	MediaCapabilities {}
mediaSession	MediaSession { metadata: null, playbackState: "none" }	MediaSession { metadata: null, playbackState: "none" }
mimeTypes	MimeTypeArray { length: 0 }	MimeTypeArray { length: 0 }
onLine	TRUE	TRUE
pdfViewerEnabled	-	-
oscpu	Linux x86_64[13]	Linux x86_64[14]
permissions	Permissions {}	Permissions {}
platform	Linux x86_64[15]	None[16]
plugins	Plugins {}	Plugins {}
product	Gecko	Gecko
productSub	20100101[17]	20100101[18]
scheduling	-	-
userAgent	Mozilla/5.0 (X11; Linux x86_64; rv:98.0) Gecko/20100101 Firefox/98.0[19]	None[20]
vendor	-	None[21]
vendorSub	-	-
webdriver	FALSE[22]	FALSE[23]

```
  const srcImg = './test-files/1.png';
  const dstImg = './test-files/2.png';
  const visual = compare.screenshots(srcImg, dstImg, (err, res) => {
    if (err) console.error(err);
    else console.log(res);
  });
})();
```

`compare.screenshots` is a function of the addon behind this Node.js application. It takes 2 parameters of the images that you want to compare. If you want to compare two JS scripts or CSS stylesheets, you'd change the comparison function from `compare.screenshots` to `compare.scripts` and the 2 parameters would change to 2 JS files or 2 CSS files that you want to compare accordingly.

Table A.4: Navigator Properties for WebKit Browsers

Navigator Property	WebKit	WebKit-None
appCodeName	Mozilla	Mozilla
appName	Netscape	Netscape
appVersion	5.0 (Macintosh; Intel Mac OS X 10_15_2) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0.4 Safari/605.1.15[24]	None
buildID	-	-
connection	-	-
cookieEnabled	TRUE	TRUE
doNotTrack	-	-
geolocation	Geolocation {getCurrentPosition: function, watchPosition: function, clearWatch: function}	Geolocation {getCurrentPosition: function, watchPosition: function, clearWatch: function}
hardwareConcurrency	8	8
ink	-	-
language	en-US	en-US
languages	['en-US']	['en-US']
maxTouchPoints	0	0
mediaCapabilities	MediaCapabilities {decodingInfo: function, encodingInfo: function}	MediaCapabilities {decodingInfo: function, encodingInfo: function}
mediaSession	MediaSession {metadata: null, playbackState: "none", setActionHandler: function, callActionHandler: function, setPositionState: function}	MediaSession {metadata: null, playbackState: "none", setActionHandler: function, callActionHandler: function, setPositionState: function}
mimeTypes	MimeTypeArray {0: MimeType, 1: MimeType, application/pdf: MimeType, text/pdf: MimeType, length: 2, item: function, namedItem: function}	MimeTypeArray {0: MimeType, 1: MimeType, application/pdf: MimeType, text/pdf: MimeType, length: 2, item: function, namedItem: function}
onLine	TRUE	TRUE
pdfViewerEnabled	-	-
oscpu	-	-
permissions	-	-
platform	Linux x86_64[25]	None[26]
plugins	Plugins {}	Plugins {}
product	Gecko	Gecko
productSub	20030107[27]	20030107[28]
scheduling	-	-
userAgent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.4 Safari/605.1.15[29]	None[30]
vendor	Apple Computer, Inc.[31]	None[32]
vendorSub	-	-
webdriver	FALSE[33]	FALSE[34]

Comparing JS Scripts

```
const compare = require('./index.js');


(async () => {
```

```
  const srcJS = "./test-files/1.js";
  const dstJS = "./test-files/2.js";
  const js = compare.scripts(srcJS, dstJS, (err, res) => {
    if (err) console.error(err);
    else console.log(res);
  });
})();
```

Comparing CSS Scripts

```
(async () => {
  const srcCSS = "./test-files/1.css";
  const dstCSS = "./test-files/2.css";
  const css = compare.scripts(srcCSS, dstCSS, (err, res) => {
    if (err) console.error(err);
    else console.log(res);
  });
})();
```

Comparing the HTML Structure

The same approach applies to comparing the HTML structure:

```
(async () => {
  const srcHtml = "./test-files/1.html";
  const dstHtml = "./test-files/2.html";
  const html = compare.structure(srcHtml, dstHtml, (err, res) => {
    if (err) console.error(err);
    else console.log(res);
  });
})();
```

It is worth noting that to see results for a compound metric as UA-Radar proposes, you can run all four comparisons at the same time.

A.6 HTML Content Comparison

The Node.js add-on behind UA-Radar uses *SFTM*[37] to compare HTML structure (just nodes of the DOM trees and their attributes). To compare the content of the nodes of the DOM trees, we used *Diff Match Patch*[38].

[37] https://github.com/lssol/sftm-csharp
[38] https://github.com/google/diff-match-patch

The complete UA-Radar comparison combines the above comparisons (HTML structure, HTML content, visual similarity, JS, and CSS).

B Exploring the geolocation impact on enforcing privacy policies

Code repository for this study can be found on GitHub[39]. It contains details concerning our crawler and the cookie consent banner detection code, along with instructions on how to run it.

B.1 Crawler Details

The directory `./crawler` contains details concerning our crawler and the cookie consent banner detection code and how to run it.

Use `./crawler/demo.py` to run the crawler, but ensure the following are in order:

1. The right Firefox profile. We use different Firefox profiles:
   - `accept` - `./data/cookie-accept`
   - `no interaction` - `./data/cookie-reject`
2. Ensure to set the right profile [40]:
   - For acceptance: `driver.install_addon('/path/to/extension/idcac.xpi')`
   - For rejection: `driver.install_addon('/path/to/extension/consent_o_matic.xpi')`
3. `./data/websites.txt` is the list of domains that we have repeatedly crawled, each 6 times from five countries.
4. When everything is ready, run the cookie consent banner detector at `./crawler/popup.py`

B.2 Results of Our Experimentation

Below is a summary of the results of our experimentation, broken down into Tables A.5, A.6, and A.7. The tables are based on interaction scenarios seen in Chapter 5: `No Interaction`, `Accept`, and `Reject`. Each table displays data across five countries: Brazil (BR), France (FR), Japan (JP), South Africa (ZA), and the United States (US).

[39] https://github.com/intumwa/cookie-prevalence/
[40] https://github.com/intumwa/cookie-prevalence/blob/929238c6326b4e9c41666ab0a96df2f2886faad5/crawler/openwpm/deploy_browsers/deploy_firefox.py#L146C80-L146C80

Table A.5: Results for No Interaction

Type	**BR**	**FR**	**JP**	**ZA**	**US**
Visited Websites with ID cookies	3272	2959	3269	3413	3384
Number of ID cookies	46047	17477	53331	60970	90680
Number of First-Party Cookies	211452	173160	231161	243408	200753
Number of Third-Party Cookies	154717	68841	184562	190757	336323
Total number of cookies	366169	242001	415723	434165	537076
Visited Websites with TP requests	3794	3788	3884	3798	3797
Visited Websites with tracking requests	2065	1462	2002	2344	2221
Number of TP requests	273352	281190	299351	323953	323977
Number of tracking requests	123763	114760	153136	160642	169911
Total number of HTTP requests	694985	642300	727550	763214	804479

Table A.6: Results for Accept

Type	**BR**	**FR**	**JP**	**ZA**	**US**
Visited Websites with ID cookies	3417	3129	3269	3333	3182
Number of ID cookies	43257	19637	54715	75392	79695
Number of First-Party Cookies	64546	57320	65771	67649	63117
Number of Third-Party Cookies	45970	18576	44664	50584	73773
Total number of cookies	110516	75896	110435	118233	136890
Visited Websites with TP requests	3850	3788	3884	3802	3828
Visited Websites with tracking requests	2476	1594	2051	2692	2447
Number of TP requests	287175	289981	303991	331462	325236
Number of tracking requests	150336	137527	161578	167894	177761
Total number of HTTP requests	768951	696623	854123	833363	829815

Table A.7: Results for Reject

Type	**BR**	**FR**	**JP**	**ZA**	**US**
Visited Websites with ID cookies	3227	2900	2911	3329	3273
Number of ID cookies	42522	17908	56118	75139	88322
Number of First-Party Cookies	66996	58994	64219	69364	75773
Number of Third-Party Cookies	43888	18494	49808	44419	95901
Total number of cookies	110884	77488	114027	113783	171674
Visited Websites with TP requests	3794	3785	3883	3804	3826
Visited Websites with tracking requests	2297	1510	2021	2552	2388
Number of TP requests	286636	283201	302909	328516	34648
Number of tracking requests	144173	119165	159270	163284	175623
Total number of HTTP requests	743016	686514	748634	776973	829051