Imperial College London
Department of Computing

Crawling the web to identify the perseverance of cookie banners and respect for choice

by
Philippe Paquin-Hirtle (PPH)

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing of Imperial College London

September 2023

Abstract

This project crawls 10,000 websites from the UK. The aim of this data collection is to determine how efficient the Brave, Firefox and Ghostery browsers are at hiding cookie banners. By answering this question, this paper will also shed light on the different techniques used by browsers to block cookie banners. To do so, a banner detection algorithm which detects the presence of a cookie banner on a website and assesses its visibility has been created. Additional privacy-related data, such as total storage and requests, are being collected and analysed to assess the overall privacy gains of these browsers when the cookie banner blocking features are enabled.

This project finds that all three browsers are able to reduce the number of visible cookie banners when compared to the Google Chrome browser. Brave is the browser with the best performance as it reduces by 86.9% the number of visible cookie banners. In terms of browser storage (cookies and local storage), every browser reduces the total number of storage units as well as third-party storage units. Brave is again the browser that reduces those values the most, with a 78.5% reduction of total storage compared to Google Chrome. It is followed by Ghostery at 67.1% and Firefox at 18.4%. As for requests, both Brave and Ghostery have been found to reduce the overall number of requests and the number of third-party requests.

Finally, a comparison between crawl data collected in the UK and in the US shows that US websites have fewer visible cookie banners. For Google Chrome, the number of storage values, especially from third parties, increases from the US vantage point. Only, Ghostery is able to limit this increase, and Brave is able to almost entirely cancel this increase in storage values.

Acknowledgements

I want to thank Dr Hamed Haddadi, from Imperial College London, and Dr Peter Snyder, from Brave Software, whose guidance has been invaluable to the success of this project.

I would also like to extend my thanks to my family and friends who have supported me throughout this project.

Table of Contents

1 Introduction

Cookies are blocks of data that are stored on a user's device. They are a popular type of client-side storage. They allow websites to remember information about users and their interactions, and can be used to implement essential functionalities such as saving site preferences, keeping a logged-in state, and implementing shopping cart features (1). Aside from implementing essential functionalities, client-side storage can also be used to track internet users. Indeed, cookies are one of the most widely used tracking technologies (2-4) and can be used to piece together the majority of a user's browsing history (5).

Overall, about 90% of websites contain at least one sort of tracking mechanism (2,5-8), meaning unprotected users are almost constantly being tracked on the internet. These trackers have a serious impact on user privacy. Collected user data is often shared across a network of third parties (3,5,9), and it can be very hard for users to know who has access to their data, or to what end that data is being used (3,5). Moreover, trackers can have serious security implications, as it has been shown that a majority of tracking information is not transferred over encrypted connections (4) and that sensitive, personal, information can be leaked (5,10). Trackers can also be used for surveillance purposes by organisations such as the NSA (4,10).

The widespread use of trackers on the internet, and the considerable implications it has on user privacy, has led policymakers to start legislating the field. Perhaps the most influential regulation passed so far is the General Data Protection Regulation (GDPR), a privacy and data protection regulation in the European Union (EU). When it came into effect in May 2018, it had a global impact (6). It is worth noting that in 2018, the UK passed its Data Protection Law, which is an equivalent regulation to GDPR.

The main impact of GDPR is that it requires consent from users before they can be tracked. GDPR defines consent as 'any freely given, specific, informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her' (art 4 (11)). There are two key nuances in GDPR's definition of consent. The first one is that it must be 'freely given'. This implies that the user must have the necessary control to make their decision (e.g. being able to refuse). It also implies that their decision should be made free of any influence (12). The second one is that consent must be 'unambiguous' and made through a 'statement' or 'clear affirmative action', which implies that consent must be explicit (6). This was meant to stop the use of implicit consent, which often interpreted the act of using a website as consenting to this website's practices (6).

Internet users in the EU will have been affected by GDPR through the now widespread use of cookie banners. Cookie banners, sometimes called cookie consent notices, are the boxes that often appear on websites to inform users about the use of cookies and to ask the user to consent to the use of cookies (Figure 1 shows an example of a cookie banner). These cookie banners can now be found on about 60% of top EU websites (12). User fatigue and cookie banner design patterns nudge users towards consenting to the use of cookies (6,13-15), without really considering the privacy implications, as this decision is seen as the path of least resistance.

To reduce user fatigue and protect user privacy, some browsers such as Brave, Ghostery, and Firefox now offer users a way to enforce their preferences while being considerably less bothered during browsing. The aim of this project is to collect and analyse data in order to determine how efficient these different browsers are at hiding cookie banners. By answering this question, this paper will also shed light on the different techniques used by browsers to block cookie banners. To the best of our knowledge, a measurement focused on the efficacy of browsers' cookie banner-blocking features hasn't been done before.

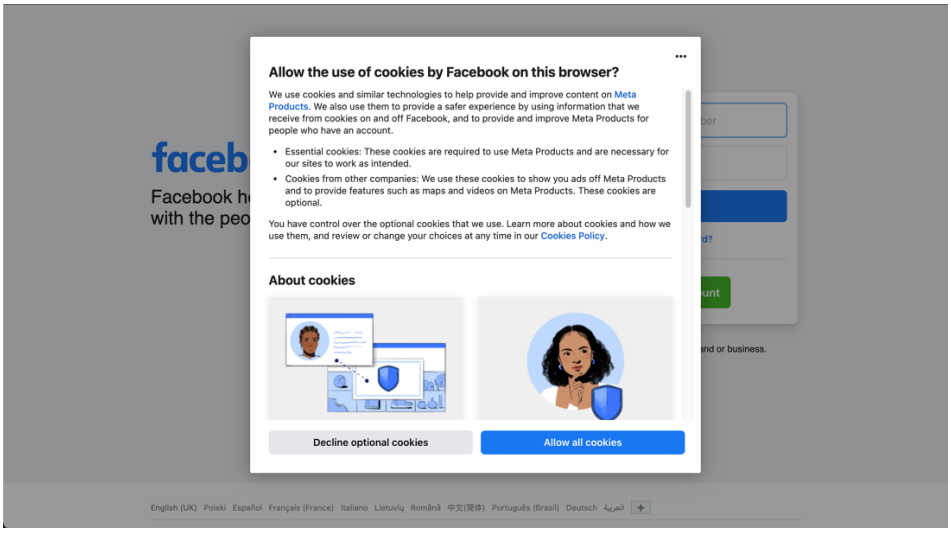


Figure 1 Cookie Banner Example

This paper will also compare these browsers against specific privacy-related metrics, such as the amount of total storage, the number of requests being made and the prevalence of third-party in both of them. This will complete the cookie banner blocking performance angle by gauging how privacy-preserving they are based on those metrics.

2 Background and Related Work

Section 2.1 provides some background on website technologies and cookies. Section 2.2 describes different cookie banner designs and how they impact users. Section 2.3 goes over previous research that looked to understand how GDPR impacted cookies and users. Section 2.4 talks about research done on counter-tracking tools. Finally, Section 2.5 explores the methods used by previous, similar, research.

2.1 Website Technologies Background

2.1.1 Cookies

Cookies can be created through two main mechanisms. The first one is using the Set-Cookie header in an HTTP response, and the second way is through scripts (3). Scripts that are embedded on the web page can make API calls to a source location (often third-party) to set cookies. Once cookies have been set, they are attached to every HTTP request made to the party that created it. The same origin policy ensures that cookies can only be accessed by, or shared with, the domain that created them.

Cookies are versatile and can be used for different functionalities. Below is an attempt to classify them without going into too much detail. A good starting point to classify cookies is to consider them to be either session cookies or persistent cookies (14). This classification looks at the lifetime and usage of the cookie. After that, more will be said about the source of the cookies.

Session cookies refer to cookies that store information only for the duration of one browsing session. This means that those temporary cookies get automatically deleted when the web browser closes. (In reality, many modern browsers are now able to restore session cookies (16), but that is ignored in this context since this is a browser property, not a cookie property.) Session cookies are also called essential cookies because they are used to provide essential website functionalities such as logged-in sessions, shopping carts, and remembering user preferences (e.g. language preference) (14). Session cookies are not used for tracking purposes. Moreover, since these cookies are essential to a website's functioning, user consent is not required for their use.

Persistent cookies, on the other hand, are cookies that do not get automatically deleted at the end of a browsing session. Instead, they remain stored until their time-to-live expires or they get manually deleted (4,14). Persistent cookies are not used to offer essential functionalities and are often considered non-essential.

A simple definition of a tracking cookie would be that the cookie must be able to identify a user (2) or a group of users sharing certain characteristics (6). Thus, the ability to remain stored across more than one session is a useful property for a tracker (4,17), though some workarounds exist (5). In practice, however, it can be hard to determine if a cookie is a tracker or not since there is no fixed list of attributes required for a cookie to be a tracker. Many researchers, such as Bielova et al. (2), Sanchez-Rola et al. (6), Merzdovnik et al. (4), Englehardt et al. (10), Trevisan et al. (17), suggest different techniques to identify tracking cookies. These will not be explained as this research does not attempt to differentiate between tracking and non-tracking cookies (a justification is provided in the results section).

Another important attribute of cookies is their source, that is, from where they are being loaded (created). Cookies can either be first-party cookies or third-party cookies. First-party cookies are cookies that are being loaded from the domain visited by the user (3). Therefore, a first-party tracking cookie can track users every time they access that particular domain or website (5). On the other hand, third-party cookies are being loaded from a different domain than the one the user directly visited (3). Third-party tracking cookies are used to track users across multiple domains (5). This is also referred to as cross-site tracking.

2.1.2 Local Storage

Local storage is similar to cookies in the sense that it is also a way to store client-side data. Local storage data is of the format key-value. It is a persistent type of storage (as opposed to sessionStorage), meaning it persists even after the browser or computer gets closed. Local storage data is accessed through an API rather than attached to every HTTP request made to that party. This means that a script by that party can create, access, modify, or delete local storage data through that API. Of course, the same origin policy is applied here, meaning that a script loaded by a certain third party can only interact with its own local storage data.

2.1.3 Third-Party Content, the Same Origin Policy, and Partitioned Storage

Without going into too much detail, it is crucial to understand the role played by third-party content on many websites. Third-party content refers to content loaded from a different origin (the combination of the protocol, domain, and port) than the website being visited (referred to as a first-party). Iframes and scripts are examples of content on a webpage that can be loaded from a third-party. Both of these can store values in client-side storage (e.g., cookies and local storage) and, therefore, store unique identifier values that can be used to track the user.

An important mechanism that needs to be mentioned, as it is critical in how content loaded from different origins can interact, is the same origin policy. The same origin policy enforces restrictions on how different web origins, all present within a single web page, can interact with each other (18). Amongst other things, it ensures that browser storage is associated with one particular origin when it is created, and that it can only be accessed by that same origin later on. This means that browser storage created by a script will be associated with that script's origin (which is the origin of the frame where the script is running, not the origin from which the script was loaded), and that this script will only be able to retrieve browser storage associated to its origin. Similarly, cookies created by an HTTP request will be associated with that request's origin, and when cookies are shared through HTTP requests, only cookies associated with the destination's origin will be shared. While this is, first and foremost, a security mechanism, it is also considerably privacy-preserving as it restricts the information third-party trackers can have access to (although some workarounds do exist) (5).

Another privacy-preserving mechanism that is being put in place by many browsers, such as Brave, Ghostery, and Firefox, is partitioned storage. Partitioned storage changes the way browsers store website data: it separates browser storage into smaller containers and assigns every website to its own specific container (19). All the data being created or queried within a website stays within that website's container. By using storage partitioning, browsers are aiming to eliminate trackers' cross-site tracking abilities. Indeed, a tracker present on multiple websites will no longer be able to access the data it stored on other websites (such as a user identifier), as data is now stored in isolated containers. Therefore, every website visited by the user which has this tracker will now have a different user identifier, as the tracker is no longer able to sync those due to the partitioning. Unfortunately, as with the same origin policy, some workaround exists. Randall et al. (20) call this workaround "UID smuggling", where UID stands for user identifiers. UID smuggling refers to inserting UIDs in navigation (HTTP) requests shared across first-party boundaries, thus helping trackers regain their cross-site tracking ability. This will be relevant to keep in mind when talking about the measurements and the results, as UID smuggling implies that HTTP requests can have serious implications on user privacy.

2.2 Cookie Banners

This section pivots slightly away from browser storage to look at cookie banners. These banners are found on a majority of websites (12). Therefore, it is likely that most, if not all, web users have previously encountered cookie banners when surfing the internet. Section 2.2.1

describes how users perceive and interact with banners, and Section 2.2.2 describes the different types of cookie banners.

2.2.1 User Perception

Previous research by Kulyk et al. (13) looked into how users perceive cookie banners. Unsurprisingly, their results show that users find that banners are disruptive, a nuisance, and an annoyance. In addition to being annoyed, users are also habituated to seeing cookie banners (13,14). That is to say that many users report ignoring cookie banners as they have gotten used to seeing them on most websites (13). When banners require an action (as opposed to a simple disclaimer), this habituation can lead users to mindlessly click on the "Accept All" (or equivalent) button in order to remove the banner and continue their browsing (13,14). Often, this is because the "Accept All" button is the fastest option which leads to the utilisation of the website (Figure 2 is an example of such a banner). Indeed, previous work (6) found that a mere 4% of websites had a clear "Reject All" option that made opting out easy.

It isn't a coincidence that cookie banners usually display a clear "Accept All" button but not a clear "Reject All" button. That is partly because of what behavioural economics calls nudging: simple changes to how a situation is presented can greatly influence decision-making (14). Indeed, the desire to take the path of least resistance and to stick with the status quo is often the preferred behaviour (14). The "Accept All" button has become the status quo, as the absence of a "Reject All" button means that to reject cookies a user must take decisive actions.

The impact of not having a "Reject All" button has been quantified by Nouwens et al. (15): it increases the consent rate by about 22%. Arguably, the observed behaviour described above shows that cookie banners do not serve their purpose very well since users act to remove a nuisance rather than make an informed privacy decision (13,14).

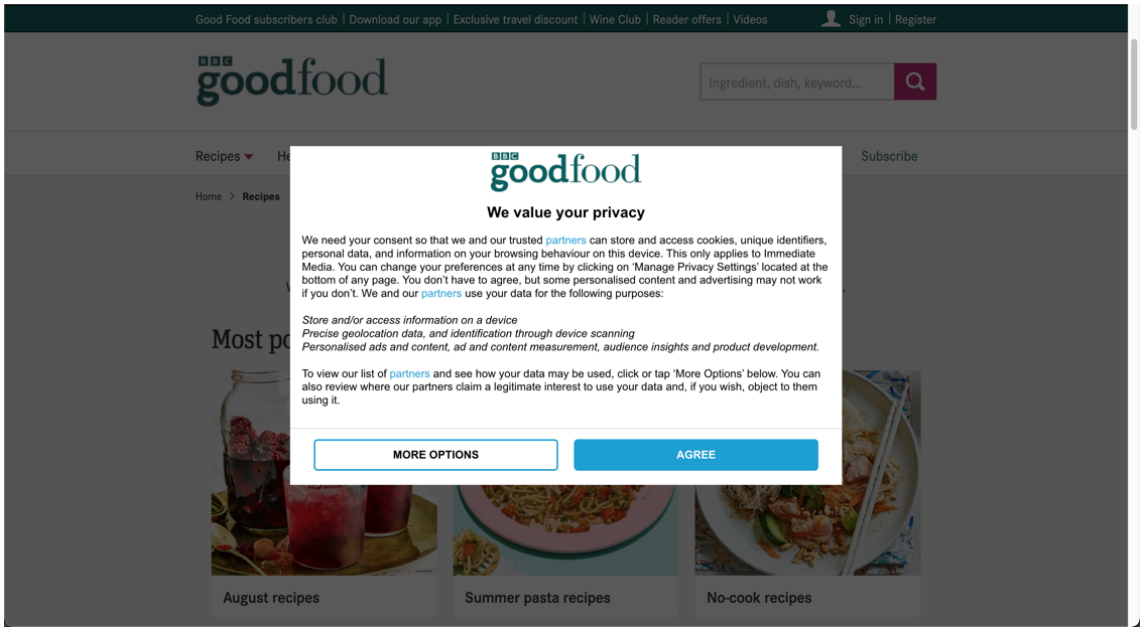


Figure 2 Example of an overlay cookie banner with a clear "Agree" button, but no "Reject" button

2.2.2 Classification of Cookie Banner Design

To understand cookie banners better, it is helpful to categorise them based on common characteristics. Previous research (6,12), published after GDPR came into effect, has suggested ways to categorise cookie banners. Cookie banners differ on two main factors: the way the banner is displayed, and the way it presents its options.

The first attribute is quite simple. There are two different ways cookie banners usually appear. First, cookie banners can appear as an overlay (Figure 2). That is to say that the user must interact with the cookie banner in order to gain access to the website. In other words,

access to the website is initially blocked (6,12). Second, cookie banners can appear, well, as a banner (Figure 3). The main difference is that banners occupy only a portion of the screen and allow the user to access and interact with the web page (6,12). The use of an overlay versus a banner has a significant impact on a user's surfing experience.

The second categorization is slightly more complicated as it includes more alternatives. Previous work (6,12) suggests that there are between 5 and 7 different ways a banner can appear to the user. A simplified version of their classification goes as follows. Cookie banners can 1) offer information without taking any user consent, 2) offer a way to accept but not reject cookies, 3) offer a way to accept and reject cookies, and 4) offer more granular settings. To clarify the above classification, it refers to how the cookie banner initially appears. Oftentimes, banners will offer a way to access a more complicated and granular settings page.

Unfortunately, most cookie banners are designed to make it harder for a user to reject than to accept cookies (6). This is in large part due to dark patterns. Dark patterns refer to malicious designs influencing user behaviour to achieve a specific goal (15). Analysing such designs is important because researchers (21) have found they are being used by 57.4% of Consent Management Providers (third parties that websites often use to outsource consent management).

Previous research (6) explains clearly how dark patterns can be used in cookie banners. The user first sees a banner with a clear “Accept All” button and a smaller “Show Purposes” link. Based on what was said before, this design already influences people to consent. The subset of users who would open the more granular settings page, by clicking on “Show Purposes”, will continue to see prominent “Accept All” buttons. If any of those “Accept All” buttons are clicked at any point in the process, then consent will be registered. Otherwise, the user can edit the settings and find the less prominent “Save and continue” buttons, which sometimes lead to a second layer of settings or even an “Are you sure?” dialogue.

In brief, it is considerably harder to reject cookies than it is to accept them. One can now clearly understand why tools, such as browsers or extensions, can be very helpful. Such tools can help users make privacy-conscious choices by reducing required user input, thus fighting habituation, and by reducing user interaction with non-user-friendly (e.g., using dark patterns) banners.

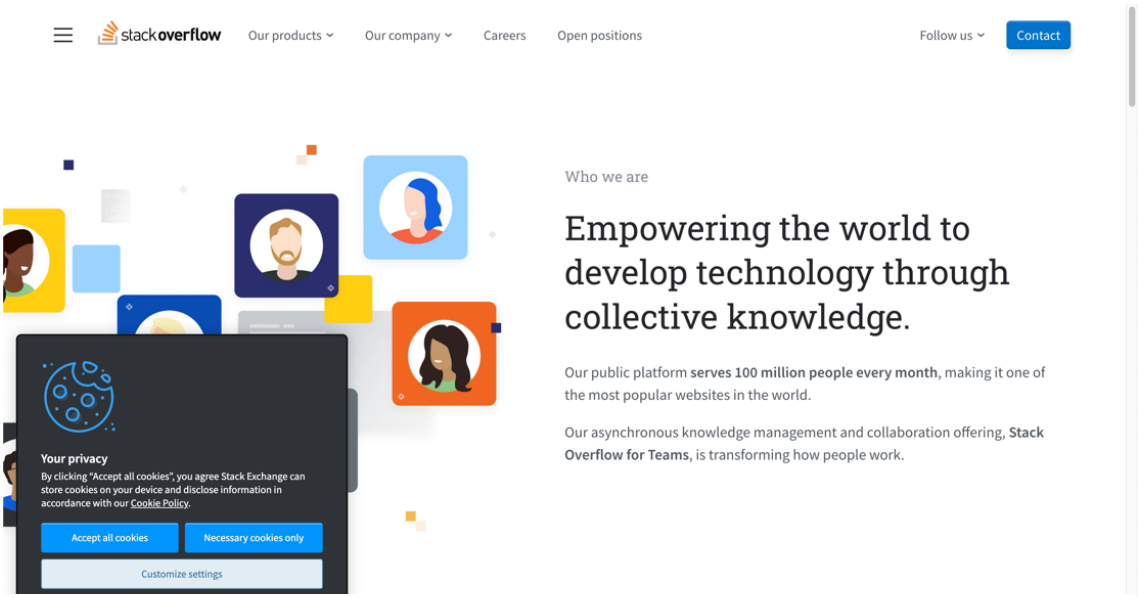


Figure 3 Example of a banner type cookie banner, with clear "Accept" and "Reject" buttons

2.3 Measuring GDPR's Impact

As mentioned in the introduction, the main impact of GDPR is that it requires consent from users before they can be tracked. The next section covers research (6,22) that studied whether or not the legislation was being respected. Section 2.3.2 draws on different research (12,23) to show how the situation has changed since GDPR came into effect.

2.3.1 Potential Legal Violations

Previous work (22) has identified four potential legal violations related to GDPR. During their study, the researchers found at least one of the four violations on more than half the websites observed.

The first violation is that some cookie banners do not offer a way to opt out from the tracking, which very clearly is against the regulation. They have found this violation to happen on 6.8% of websites (22).

The second violation is that some cookie banners have pre-selected some choices. This is a violation because pre-selected choices can influence users which means that consent is not unambiguous in this case. This one is by far the most common violation, being found on 46.5% of websites (22). Another study found that preselected options led to a considerable increase in consent from 0.16% to 83.55% (21).

The third violation is that consent is sometimes stored by default, even before the user explicitly makes a decision by interacting with the cookie banner. This violation is found on 9.9% of websites (22). This means the user could be tracked before giving explicit consent, which is a clear GDPR violation. In fact, other research (6,17) confirms that users often are tracked before consenting. The results show that this happens on 49% of websites (17) or on 92% of websites (6). These results are considerably higher than the initial violation, and this is because these papers (6,17) use a wider criteria. Instead of looking at whether consent is stored by default (as done in (22)), they look at whether tracking actually occurs before consent is given. Actually, to be more precise, they notice that tracking occurs even before the cookie banner appears (6), leaving users powerless unless they have a tool automatically blocking the trackers. The difference between the 49% result (17) and 92% result (6) might be explained by the different methods these researchers used to identify tracking cookies, as the authors of the paper with the lowest result of 49% (17) are aware their strict categorisation might lead to false negatives.

The fourth violation is the non-respect of choice, which is found in 5.3% of websites they observed (22). Non-respect of choice means that consent is stored despite a clear refusal from the user (22). As with the third violation, other research reaches a similar conclusion. By looking at the cookies being stored, rather than the consent string (which is where the user decision is stored), researchers (6) found that only 2.5% of websites removed cookies after a user refused tracking. To better understand the implications of this, recall their finding that 92% of websites track users before receiving consent (6). Combined, their results imply that around 90% of websites track users despite them opting out. Sanchez-Rola et al. (6) mention that users who never consent to cookies are still being tracked by most websites. As with the third violation, the difference in results can most likely be explained by the fact that one (22) looks at the consent being stored while the other (6) looks at the trackers being stored. A seemingly crucial nuance.

2.3.2 Comparison Pre- and Post-GDPR

As mentioned earlier, multiple papers have examined how GDPR impacted different metrics. This section covers some interesting findings that fall in that category, providing a good background for the type of measurement that will be done in this project.

First, a paper (12) looked at whether or not the proportion of websites with a cookie banner changed. Unsurprisingly, they found a 16% increase in the number of websites with a

cookie banner. With this increase, the paper found cookie banners on 62.1% of websites covered by GDPR.

Second, research (23) looked at the impact of the location variable. They set up an experiment with websites in and out of the GDPR's coverage (based on Top Level Domain) and user locations in and out of the EU. Previous work (12) explained that websites' incentives would lead websites to show cookie banners only to users under GDPR's jurisdiction. The logic is that websites and third parties should not want to offer a way to opt out of tracking, unless required to, since tracking can lead to significant monetary benefits (24). In the end, the study's results (23) found that user location had little impact but that the location of the top-level domain impacted the presence of cookie banners.

Third, is the question of whether or not GDPR had an impact on the prevalence of trackers. Unfortunately, some have shown that the prevalence of trackers has not been affected (12) and that tracking cookies can still be found on more than 90% of websites (6). It also seems that most websites have not changed their consent type, with the majority still using an opt-out mechanism (12)

In conclusion, users are not less likely to be tracked than before. On one hand, users have more control over their privacy due to an increased number of cookie banners with the ability to deny consent. On the other hand, this control does not amount to much due to multiple regulation violations and mechanisms put in place to make opting-out harder. These alarming results show the importance of anti-tracking tools and technologies. Internet users should be able to make their privacy decisions easily and trust they are being respected.

2.4 Counter-tracking Technologies

Counter-tracking technologies have an important role to play in helping the user have a more seamless web experience while enforcing their privacy preferences. Section 2.4.1 looks at previous research which focused on tools that block trackers. Section 2.4.2 provides an overview of the tools that exist to block cookie banners.

2.4.1 Blocking Trackers

The main challenge for counter-tracking technologies is to accurately understand what is or isn't a tracker. Indeed, many scripts, cookies, etc., can be essential to a website's functionality. Therefore, counter-tracking technologies must try to block trackers while keeping a website's functionality as intact as possible. Different approaches exist to classify what is or isn't a tracker, the most popular of which is probably blacklisting.

Tools that use blacklisting rely on a list of script names, URLs, and domains which are to be rejected (blacklisted) (8). Blacklists are a very straightforward approach to blocking pre-identified trackers, but their main weakness is that they cannot detect previously unseen trackers or simple changes like a name change (8).

Blacklists can be generated in different ways, and research has shown that this can impact the likeliness of a blacklist to detect a tracker. First, there are community-driven lists like AdBlock's (25) EasyList and EasyPrivacy (4). These lists are maintained by a wide forum community that helps the main authors keep the list updated (26). Second, there are centralised lists meaning the organisation behind the tool is responsible for building and maintaining the list. This category includes tools like Ghostery (27) and Disconnect (28) (4). Merzdovnik et al. (4) have found that the former option, centralised rulesets, are better at detecting and blocking trackers than community-driven rulesets.

A less frequently used alternative to blacklist tools are heuristic (algorithmic) tools. Those have a completely different approach to blocking trackers. They use heuristics to identify trackers rather than a predetermined list, meaning the tools learn and become more and more accurate over time. This category includes tools like Privacy Badger (29) (4).

In addition to the extensions mentioned above, major browsers, except Google Chrome, have been offering proactive anti-tracking features that are focused on reducing cross-site tracking (30). Indeed, privacy-focus browsers such as Brave and DuckDuckGo offer to block trackers, Firefox offers "Total Cookie Protection", and Safari offers "Intelligent Tracking Prevention" (30). Some of these browsers either use, replicate, or provide native support for some techniques found in tracker-blocking extensions, such as blacklisting (4).

It is relevant to go into some more detail about some of the key privacy features offered by default by the different browsers being studied. This is not an exhaustive list, and it is only relevant to give the reader an understanding of what type of privacy-preserving actions those browsers are taking.

Most of Brave's privacy features are gathered under the "Brave Shields" umbrella (31). Brave Shield offers features which block ads, cross-site trackers, third-party cookie tracking, and fingerprinting (a way of doing cookie-less tracking). Brave mentions being able to detect cloaking strategies used by trackers to hide themselves from blacklists, which should improve its efficacy.

As for Ghostery (32), both the browser and the extension block trackers using a blacklist of 5000 trackers and 3000 companies. Moreover, Ghostery completely intercepts requests made to known tracking servers. This is more privacy-preserving than only blocking cookies as the server gets completely cut off and cannot even collect the user's IP address. Like Brave, Ghostery explains they can detect cloaking strategies and block the trackers.

Firefox offers Enhanced Tracking Protection effort which uses a blacklist to block trackers, and more recently launched Total Cookie Protection which helps overcome the shortfalls of the blacklist approach (33). Firefox also blocks cross-site tracking as part of its settings enabled by default.

2.4.2 Blocking Cookie Banners

Some tools also exist to block cookie banners. There are interesting differences in how different tools go about blocking cookie banners. Broadly, the differences are related to how they detect the banner, how they block the banner, whether the tool interacts with the cookie banner, and, if so, what the interaction is. This section does not claim to exhaustively explain how each of the tools works but rather points out certain of their important and easily identifiable characteristics.

First are extensions such as "I don't care about cookies" (34). This extension uses CSS elements to identify cookie banners. More precisely, it identifies cookie banners by comparing CSS element names found on a web page with a list of over 9,000 names often used by cookie banners. If a cookie banner is detected, the extension automatically clicks on the "Accept All" button, thus making the banner disappear once the decision has been recorded. Obviously, this is not a privacy extension but rather one that aims to make browsing more pleasant.

Second are extensions like Consent-O-Matic (35,36), as well as browsers like Ghostery (37) and DuckDuckGo (38). Consent-O-Matic, which is used by Ghostery and DuckDuckGo, detects cookie banners rendered by Consent Management Providers (CMPs). The tools then interact with the cookie banner, but this time selecting the most privacy-preserving option (if so desired by the user) (35,38). In other words, these tools will click the "Reject All" or the closest there is to that option. The cookie banner therefore disappears because a user decision has been recorded. DuckDuckGo also explains that when no opting-out options are offered, they hide the pop-up rather than selecting one of the consent options available (38).

Third are some extensions like uBlock (39) or some browsers like Brave. Their method avoids interacting with the cookie banner (40). Rather, they identify the section of the HTML related to the banner through the use of a blacklist (EasyList Cookie (41)). They then remove the banner HTML from the HTML document, thus completely removing rather than hiding the

banner. Brave claims this is the most privacy-preserving approach as it does not involve any trust or communication with any system (40).

Fourth, Firefox also offers a feature to block cookie banners (42). This feature is currently not enabled by default but can be set in the about:config page by changing the value of "cookiebanners.service.mode" to either 1 or 2. These two options behave slightly differently: option 1 rejects all cookies when the option is available, and option 2 rejects all cookies when the option is available but falls back on the option to accept all cookies. It seems that option 1 will be the one favoured by Firefox, as that is the option they will enable if the user chooses to "reduce cookie banners" in the about:preferences page (currently only in Firefox Nightly).

As a reminder, the goal of this project is to fill the lack of research on the efficacy of such browser features, and to better understand how well these different methods protect user privacy. The above solely serves as an introduction to the reader on how different extensions and browsers work, based on the information available on their respective websites. Only, this project does not take any of the information presented above for granted. Instead, it takes appropriate measurements in order to compare the true effectiveness of these features, rather than only comparing features as was just done.

2.5 Methods

This section looks at relevant methods and measurements made in previous related research. The reasoning and decisions made by previous researchers will help inform the method and parameters to be used in the experiment for this research. Section 3.3 will address in more detail the decisions that were made for this experiment.

2.5.1 Crawl Methodology

This section focuses on the considerable number of research papers that performed some form of automatic measurement on the web (a crawl) to study various tracking-related phenomenons. More specifically, this section looks at the crawling methodology and parameters used for previous research. Having a precise set of parameters is key to obtaining reproducible results (43).

Website selection: In most cases, researchers use Alexa (3-7,13,44) or Tranco (22,30,43) as a way to determine the most popular sites which they intend to crawl. The scale (number of web pages or domains visited) is a factor that varies considerably across different research. Indeed, it has been found to range from a million or more web pages (8,9,44) to a couple of thousands (12,23,30). This is mostly due to the specific goals of each project, and therefore isn't the most informative metric for this project. Finally, Rasaii et. Al. (45) explain how they selected websites when they needed to select a subset of the Tranco top-10k websites. They select three tiers of websites (top, middle, and bottom) by choosing the top-100, 1001–1100, and 9901–10k websites. These numbers are specific to the fact that they wanted to achieve a sample of 300 websites, but could be generalizable to any sample size desired. They justify such a sampling technique by explaining that one cannot be certain that website characteristics and behaviours are the same on high- and low-traffic websites. In other words, it is their attempt to reduce sampling bias.

Timeout time: The timeout value is the maximum time the crawler should wait for the website to finish loading. If the crawler times out, it usually moves on to the following site. Timeout values are generally in the range of either 90 sec (17,44), 60 sec (8,10), or 30 sec (30,43). In one case, the timeout value was considerably lower, at 10 seconds, but the researchers repeated the attempt three times (22). Other research (43) also uses repetition to combat noise.

Dwell time: The dwell time represents the time the crawler remains on a web page. Dwell time allows to ensure trackers that are installed after a few seconds are still captured in the

experiment. Previous work has used a dwell time of 15 sec (8,43) and 30 sec (30). Moreover, some have tested different dwell times between 15 and 60 sec in 5 sec increments and found no variations (43).

Temporal variations and Parallelism: Because web pages contain many variables that can change over time, such as ad campaigns, for example, researchers perform their crawl in a parallel manner since this ensures the conditions of the crawls are as standardised as possible (4,23). Since parallelism reduces the time required to perform a crawl, many researchers use parallelism to help with the scale of the crawl (4,7,17,44).

Vantage point: The crawl's location (source IP) can impact website behaviour and crawl results (12,43). Research found that it is best to use residential or academic IP addresses and to avoid cloud servers like AWS (43).

Headful state: Using a headless crawler can also affect results due to bot detection mechanisms (30). Therefore, it is recommended to use non-headless, or headful, crawlers (30,43,44).

2.5.2 Detecting Cookie Banners During an Automated Crawl

This section presents the methodology four groups of researchers used to automatically detect cookie banners during a web.

Kampanos and Shahandashti (46) and van Eijk et al. (23) found that they could detect cookie banners by using the list of CSS elements used by the extension "I don't care about cookies" to detect cookie banners. Importantly, van Eijk et al. (23) point out that this technique can lead to false positives. Kampanos and Shahandashti (46) take additional steps to limit their rate of false positives by checking that the HTML returned by their crawler is of reasonable length and that the words "cookie" or "cookies" can be found at least once in the HTML.

Matte et al. (22) created a browser extension, Cookie Glasses (47), that detects cookie banners that are implemented by following the Transparency and Consent Framework (TCF), a framework that standardises the collection and exchange of consent data. Their extension only detects TCF banners because they detect them based on precise implementation details that TCF banners must follow. They detected TCF banners on 13.6% of the top-1000 websites with the ".com" top-level domain, and on 6.2% of all the websites they crawled.

Rasaii et al. (45) propose their own cookie banner detection algorithm, which is not dependent on any third-party (e.g. CSS list) or third-party attribute (TCF compliance). They randomly sampled 50 websites of the Tranco top-100 list and came up with a corpus of eight words often found in cookie banners. The eight words that make up their corpus are: "cookies", "privacy", "policy", "consent", "accept", "agree", "personalized", and "legitimate interest". They also proceeded to translate these words into eleven different languages. Once they find a word match using their corpus, they move up in the DOM tree to find the largest div containing only the cookie banner. They identify this element by looking for a positive z-index or a fixed position attribute. From there, they move down the DOM to find the smallest div that contains all of the banner's elements. After manually inspecting the Tranco top-1k websites, their results show that their technique detects cookie banners with 99% accuracy and low false positive and false negative rates.

3 Web Crawler Implementation

3.1 Browser Selection Justification

The main goal of this study is to compare the performance of different browsers when it comes to blocking cookie banners and helping to preserve user privacy. With this in mind, five browsers had been identified as candidates for the measurement at the start of the project. These browsers were Google Chrome, Brave, Firefox, Ghostery, and DuckDuckGo. Google Chrome was chosen as the control browser as it does not have a cookie banner blocking feature. It also has very few privacy settings enabled by default. On the other hand, Brave, Firefox, Ghostery, and DuckDuckGo all offer privacy-preserving and banner-blocking features (often by default), as mentioned in Sections 2.4.1 and 2.4.2. Moreover, given all browsers implement their cookie banner blocking feature slightly differently, it made them interesting picks for this study.

From that initial list, DuckDuckGo was removed from the experiment because it uses WebKit, which is incompatible with Puppeteer, the tool used to perform the web crawl. The backup plan was to use the DuckDuckGo browser extension to be a proxy for the DuckDuckGo browser, but unfortunately, the DuckDuckGo extension does not currently have the functionality of blocking the cookie banners, as opposed to its branded browser. Due to time constraints, it was decided to exclude DuckDuckGo rather than use a second tool to perform a DuckDuckGo crawl. Moreover, the final crawler ended up using the Ghostery extension mounted on a Google Chrome browser. This decision was made solely for implementation purposes, and is expected to have little to no impact on the results due to the extension and browser functioning almost identically (the browser integrates the extension).

3.2 Puppeteer

This project uses Puppeteer (48) to perform the measurement. Puppeteer is a Node library made to help automate tasks, such as web scraping and web crawling. It provides an easy-to-use API to control Chrome and Chromium (e.g. Brave) browsers. A good portion of that same API currently supports the Firefox browser as well (and by extension, the Ghostery-branded browser, as it is built on top of Firefox). As described later, some adaptations had to be made when faced with non-compatible features for Firefox.

3.3 Crawl Parameters

Section 2.5.1 exposed relevant crawl parameters chosen by previous research. This section exposes the choices that were made for this project. Note that the code repository for this project is available here: https://github.com/PhilippePH/cookie-banners.

Please note that two complete UK crawls have been performed during the project, the reason for which is described later in Section 4.1. This is mentioned now because some parameters have been changed between the crawls, which is relevant to the present section. The values that have changed and the justification for the changes will be mentioned below. Also, note that the US crawl uses the same parameters as the second UK crawl.

Vantage point: The crawls were made either from a UK or a US vantage point.

Website selection: In total, 10,000 websites were visited by each of the browsers from the UK vantage point. The first half of those (5,000) were also visited by each of the browsers from the US vantage point. The list of websites was made using the Tranco top-100k list, downloaded on August 9th. From that list, the top-1k websites were selected to be the first 1,000 websites crawled in order to have a good representation of the most visited websites. Then, the remaining 99,000 websites were divided into three tranches (1,001-33,000 / 33,001-67,000 / 67,001-100,000). The websites in each tranche were shuffled. Then, the first website of each tranche was added to the list of websites to crawl until reaching the desired number of websites.

This approach ensured the crawl visited websites across the popularity distribution, therefore limiting the data collection bias.

Navigation timeout: The first crawl used a 10-second navigation timeout, with the load event as a load confirmation. This means the crawler waits for up to 10 seconds to receive the load event. If the load event is not encountered in that time frame, it times out, and the crawler moves on to the next website. Every website that times out is re-crawled up to two other times after the completion of the crawl. This is an attempt to limit the impact of internet unpredictibilty, and thus to convert a timed-out website to a successful website.

The 10-second value was initially chosen after some tests showed that most websites would either successfully load in 10 seconds or never load, no matter the navigation timeout value. This could be due to some pages never firing the load event or to Puppeteer not receiving it. That mostly happened with Firefox and Ghostery, hinting at a compatibility issue that the navigation timeout would not help fix.

The load event was used as a proxy to determine when a page had successfully loaded. Out of the different load proxies offered by Puppeteer, this one was chosen because it reduced the number of timeouts during testing. This event is also fired later than the DomContentLoaded alternative, which is good as it maximises the number of resources being loaded before starting the data gathering, which, in turn, maximises the chances of observing the cookie banner's final state (present or blocked).

For the second UK crawl, the navigation timeout was modified to 15. While the points above remain, the considerably high timeout values warranted a pivot. Probably due to the use of parallelism and prolonged utilisation time, the machine running the crawler (and, therefore, the browsers) was slower than in the test environment. Therefore, the switch to a 15-second timeout with a second re-try was made to try and accommodate this. Overall, a total of up to 30 seconds attempting to load the website is still present.

Data collection timeouts: 5-second timeouts were also set when interacting with the page console to collect data. This was implemented in response to a flaky (non-reproducible) behaviour where evaluating window.origin would never complete, and the crawler would remain stuck. This very disruptive behaviour was rarely observed during testing, was not reproducible, and seemed to be caused by many very specific edge cases. It was determined that the best course of action was to set a timeout and risk not collecting every data point for every website.

Other timeouts: For the second UK crawl, an overarching 30-second timeout has been added. This timeout englobes everything related to one page, from loading to taking the measurements to adding the data to the database. Timeouts were also added to the methods in charge of closing a page (1-second timeout) and closing the browser instance (5-second timeout). These changes were made to eliminate some behaviour observed in the first crawl where the crawler would remain stuck in a place where no timeouts had been set.

Temporal variations and parallelism: During the first UK crawl, four crawlers were launched in parallel to ensure websites are visited as closely as possible to reduce the effect of the variability of websites. Each crawler was allowed to progress at its own rhythm. This means that despite being launched simultaneously, the Firefox and Ghostery crawlers took an extra day to complete the 10,000 website crawl. This meant that the temporal variations were limited to the changes that could arise in one day. This was judged to be strict enough to ensure results were comparable.

For the second crawl, a few changes were made. First, the Ghostery extension mounted on a Google Chrome browser was used instead of the Ghostery-branded browser. This is because the Ghostery browser uses the Ghostery extension by default, meaning they are expected to generate the same results. However, doing so meant that the Ghostery measurement would use a fully supported Google Chrome rather than the less well-performing Firefox

(again, the Ghostery branded browser is built on top of Firefox). Second, the parallelism technique was tweaked. Since the Brave, Ghostery, and Google Chrome browsers were progressing at a similar pace during testing, the three were launched in parallel. It takes those browsers about one day to complete the 10,000 website crawl. Once those browsers have completed their crawl, three identical instances of the Firefox browser are launched in parallel. This allows the Firefox crawl to be over more quickly, as each instance is responsible for one-third of the websites. It also takes about a day hours for Firefox to complete the crawl. The Firefox instances could not be launched in parallel to the three other browsers because six browser instances running in parallel would have overwhelmed the machine on which the crawl was being performed.

Bot detection: Multiple crawl parameters were set in order to avoid bot detection. In addition to following recommendations from previous research (43), Sannysoft (Antibot) (49) was used to test how well the crawler was evading different bot detection techniques.

The crawler ran in headful mode, with its default viewport maximised (which makes the page take the whole window size).

The puppeteer-extra-plugin-stealth (50) plugin was also used. It allowed Google Chrome (including when it is used with the Ghostery extension) and Brave to pass all the bot detection tests. The compatibility with Firefox (and the Ghostery branded browser) is incomplete, so the extension was not used. The only bot detection test left to pass for Firefox was the webdriver flag. The webdriver flag indicates that a browser is being controlled by a bot, which can lead to websites changing their behaviours or asking for CAPTCHA-like challenges before allowing access to the website. Section 5.1.2 describes the multiple attempts to modify or delete this flag, but they were unsuccessful. The Firefox and Ghostery-branded (when used) browsers are thus crawling with the webdriver flag set to true, which is a limitation.

Jueckstock et al. (43) have shown that residential IP addresses were less likely to lead to a change in the behaviour of websites. Therefore, initial attempts, described in Section 5.1.1, used proxy servers with a residential IP address to crawl from. After the set-up evolved to its current state (described in the next section), the UK crawler was launched from a residential IP address (without requiring a proxy), and the US crawler was launched using a VPN service. The limitation of using a VPN service is explored in Section 5.1.1.

Browser parameters: The Google Chrome browser instance runs with all its default parameters. The Brave browser instance runs with all its default parameters, and when prompted upon the first website visit, the cookie banner blocking setting was enabled (which enables EasyListCookies in brave://settings/shields/filters). Similarly, for the Ghostery browser instance, the Ghostery privacy settings were activated when prompted upon opening the browser, or extension, for the first time. Moreover, upon the first website visit, the NeverConsent feature was activated for all websites. Otherwise, all the settings are the default options. Finally, one setting has been changed from its default value for the Firefox browser instance. In about:config, the setting "cookiebanners.service.mode" is set to 1. This enables Firefox's cookie banner blocking capability. As mentioned previously, this seems to be the setting favoured by Firefox as it is the mode enabled when activating the feature in Firefox Nightly (42), and it is also the most privacy-preserving one, which fits the goal of this project.

3.4 Experiment Set Up

The crawl was performed from a MacBookPro (hereafter referred to as the server). The server also hosted a database where most of the data was being. When not using a VPN, the server was accessed and controlled through an SSH connection, as its graphical interface becomes unusable when the crawler is running. An attempt was made to set up a reverse SSH

tunnel to continue using SSH while the server was connected to a VPN but was abandoned as it was not worth the struggle.

This set-up was established relatively late in the project, after needing to pivot from the idea of using residential proxies to do the crawl. This pivot and the previous set-up are explained in more detail in Section 5.1.1.

3.5 Data Collection

First, request and response data are being collected, using Puppeteer's API, which offers a straightforward way to intercept them. The origin of the frame making the request and the URL of the requested resource are being gathered for the requests. The frame origin is used to identify the party making the request. As for the response data, the URL of the responder is being collected, the content type and content length headers.

Second, storage data is being collected. This includes cookie data as well as local storage data. In both cases, the frame origin is being collected. For cookies, all the cookies set in this frame are collected using Puppeteer's built-in method ".cookies()". Then, the frame origin, cookie name (key), cookie value, and cookie domain (specific field in the cookie) are added to the database. For local storage, the frame origin and all the name (key) value pairs found in a frame's local storage are added to the database.

The storage data collection is done by recursively visiting every frame present on the webpage. That is because of the same origin policy. Indeed, as mentioned in section 2.1.3, the same origin policy implies that storage can only be accessed from within the origin associated with it. The recursive approach ensures that every frame on the page is visited, thus allowing the crawler to collect all of the storage data for a website.

Finally, for the first crawl, the HTML of the webpage was downloaded in its entirety after the load event had been fired. This maximised the chances that all the content had loaded, including, most importantly, the cookie banner (which sometimes takes a few seconds to appear on the page). The reason why this was removed in the second crawl is linked to the cookie banner detection algorithm, which is explained in the next section.

3.6 Cookie Banner Detection Algorithm

One of the data points being analysed as part of this project is the efficiency of different tools at blocking a cookie banner when a banner is present on the webpage. This means the banner detection algorithm must be able to detect the presence of a cookie banner and, if found, evaluate whether the cookie banner is visible or has been hidden somehow. One thing the algorithm is not concerned about is to clearly identify the borders of the cookie banners (meaning, all the divs that make up the cookie banner). As long as the algorithm can correctly assess the visibility of the cookie banner, it has completed its goal.

The algorithm also needs to recognize various formats of banners. Indeed, the crawler is visiting pages regardless of their top-level domain, and it is doing so from a European and a non-European vantage point. This means that techniques leveraging specific attributes of Consent Management Providers or frameworks (like the TCF) might not be effective.

The algorithm below was tested on the top-250 websites. Ground truth values were collected manually for these 250 websites from the four different browsers being studied. It was decided to use 250 websites as that struck a good balance between maximising the sample size and keeping a reasonable total number of websites to manually visit (1,000). The top-250 websites were chosen as this subset was likely to maximise the proportion of sites with cookie banners and, therefore, maximise the training data for the algorithm for a given sample size. While every algorithm iteration was tested with those top-250 websites, the results of every test for every iteration will not be presented for conciseness. Section 3.6.5 will describe the

accuracy of the last iteration of the algorithm, and Section 4.2 will present its results. A visual representation of the final version of the algorithm is provided in Appendix 2.

3.6.1 Corpus Creation

The algorithm uses word matching to detect the cookie banner on the web page. Therefore, before describing the different iterations of the algorithm, it is worth explaining how the different corpuses that were tested were created. The list of search terms included in every corpus is shown in Appendix 1.

To build the first corpus, an analysis of the text of the top 50 banners was made. A simple Python program counted the number of times words were repeated in those cookie banners. From this list, a manual check was made to see which words could be relevant. This meant the words had to be specific enough, as the goal was to distinguish the banner from the other text on a webpage. During the manual check, special attention was kept to words that fit into word categories often found in cookie banners. The categories that helped guide the search were: “consent”, “non-consent”, “type of cookies”, “settings”, “information”, “policy”, “third-party”, and “other”. This is not central to the creation of the corpuses, and therefore for conciseness, these categories will not be explained further. Finally, the words also had to have been found in a large enough number of distinct cookie banners. The result of this initial attempt is the “Long Corpus”.

Then, testing was made with the “Long Corpus” to see how it performed. The output of these tests gave the accuracy and the words used to find the banner. From there, the “Medium Corpus” was created by keeping the least amount of words from the “Long Corpus” that would preserve the same detection accuracy. The motivation for this shorter corpus was that a more concise corpus could help reduce the number of false positives.

In addition to the corpuses generated above, the corpus from Rasaii et al. (45) was also tested. This is the corpus called “Exploring The Cookieverse Corpus”. This corpus is used as part of their cookie banner detection algorithm, which achieves very high accuracy, which is why it was worth testing its performance with this algorithm.

Seeing how short the “Exploring The Cookieverse Corpus” was, it was decided to generate a corpus similar in length using the most useful words found by the Python script and the other corpuses’ tests.

After this initial round of corpus creation and tests, the corpuses were modified a few other times, but that will be described in more detail in the following sections. It quickly became apparent that the medium corpus was the one performing best, and towards the last few iterations, it was the only corpus being tested. The final version of the corpus used is described in Section 3.6.5.

3.6.2 First Iteration Batch

The first iteration of the algorithm combined the use of a threshold and of a corpus of words related to cookie banners. This iteration aimed to develop an efficient technique to detect the presence of a cookie banner. It did not yet focus on evaluating its visibility. This iteration was tested on the downloaded HTML of the top-250 websites visited by the Google Chrome browser (as this is the “control” browser).

The main idea is that the cookie banner sub-tree contains a small percentage of the nodes of a website's document object model (DOM). Therefore, using a certain threshold value (1%, 5%, and 10% of the total number of elements were the values tested) gave a maximum sub-tree size. Then, one could create a list of sub-trees to test based on the condition that they needed to reach terminal nodes within the number of elements threshold (in a breadth-first manner). This means that the threshold was used to eliminate all elements that were too high in the DOM, and only kept sub-trees of a certain size that reached terminal nodes.

Then, every sub-tree was submitted to a word search. The word search was performed on all elements of the sub-tree, which had a text attribute. The element within the sub-tree with

the highest number of word matches was selected as a cookie banner candidate. Then, after every sub-tree had been tested with this word search, the cookie banner element was chosen amongst all the element candidates by selecting the element with the highest number of word matches.

In this iteration, a grid search method was used to test all the combinations of threshold values and corpuses to determine which pair of parameters produced the best results. This was especially useful in the first iteration because it required making the least number of assumptions about the parameters. Further iteration didn't use a grid search; once certain combinations were found, progress was made by tweaking one parameter at a time.

While this iteration of the algorithm is several steps away from the final version, it played a significant role in helping orient the algorithm's development's efforts and gain a deeper understanding of how the algorithm interacted with banners. As a temporal indication, after the first iteration of the algorithm, which worked on offline HTML data, it was decided to launch the first UK crawl as all of the other measurements were ready. The first UK crawl downloaded the HTML file of every website crawled so that they could be used by the algorithm. When trying to add the visibility detection feature, while the first crawl was running, it became clear that it would need to be done online rather than offline, and that a second crawl would be required.

3.6.3 Second Iteration Batch

After that initial testing, a second batch of iterations was made. This iteration differs from the previous one in three main aspects: it uses a fixed value threshold rather than a percentage threshold, uses a sub-tree of elements rather than only considering one element, and assesses the cookie banner's visibility.

First, it was decided to change the percentage threshold idea for a certain, fixed, size. This is because websites have varying DOM sizes, meaning the number of elements considered varied considerably when using a percentage threshold. This was found not to reflect the reality: cookie banner sizes do not vary in accordance with the DOM size, they are relatively constant in terms of the number of elements they contain. Thus, the pivot was to use a fixed value of elements that would be considered. In other words, rather than considering elements up to a certain percentage away from end nodes, they would be considered up to a certain number away from the end nodes. Combined with the information gained with the tests using a threshold, a handful of cookie banners were analysed to determine an appropriate range for that value. It was determined that going up approximately five levels from a terminal node would in most cases encompass most of the cookie banner. Some tests were made by varying this value from 3 to 7, but that did not yield a large change in results. This is mostly due to the fact that the algorithm does not need to ensure it has found all of the elements of the cookie banner; it must simply ensure that what it has found is a cookie banner.

Also, it was decided to add a condition to the algorithm to limit noise and the total number of sub-trees created. This condition stated that the algorithm could only start the creation of sub-trees from terminal nodes which also had a word match. This condition seemed promising since cookie banners are usually at the end of their branch. When implemented, this measure helped reduce the number of false positives compared to the first iteration. It more easily ignored scripts setting cookies or text in the webpage that had matches with words in the corpus.

Another main modification is that the algorithm would now create sub-trees composed of all the elements between that terminal node and the highest parent considered, rather than consider only one element. Therefore, all children of the highest parent were recursively added to the sub-tree. There was a worry, however, that using a fixed value of parents would lead the algorithm too high up the DOM in certain websites. Especially if the cookie banner's only parent is the <body> element, then the whole of the HTML page would be included in the sub-

tree, which would lead to misidentification of the cookie banners and possibly many false positives. In order to avoid that pitfall, a limit on the number of children any parent had was put in place. This mechanism ensured that if a sub-tree grew larger than the threshold, it would stop growing before adding that parent to it. Threshold values of 5-10-15-25-50 were all tested. Higher values saw a decrease in accuracy due to the problem explained above. On the other hand, smaller values led to a small increase in false negatives due to the exclusion of elements that were part of the banner but had too many children.

This change of switching from considering a single element to a subtree was implemented for two reasons. The first one is that this makes the cookie banner sub-tree more likely to stand out, as matches from its multiple elements all contribute to this sub-tree being chosen instead of another sub-tree. When it was compared on a per-element basis, there were chances that an element that was not part of the banner had enough word matches to win over a banner element. With this pooling of results, this becomes less likely as the corpus has been crafted to generate multiple hits with the banner sub-tree. The second reason is for the visibility assessment. Before explaining why, explaining how the visibility is being assessed is useful.

The algorithm's first part, which includes the sub-tree creation and selection, is all made within the browser context (i.e., within a page.evaluate(), meaning it is all executing in the browser console). Only, the cookie banner's visibility is evaluated in the NodeJS environment using the Puppeteer API. The most efficient way to return from the browser context to the NodeJS context and return something that would remain defined (returning an element (JSHandle) object becomes undefined once outside of the browser context) was to return the element's class name. Indeed, Puppeteer is able to access elements using attributes such as the class name, and the value returned can be used by other methods offered by the Puppeteer API. Thus, after re-accessing the element from the NodeJS context, the very appropriate Puppeteer isVisible() method was used to check if the element on which it is called is visible. The isVisible() method gave very good results when it was tested with the four browsers and their various methods of blocking cookie banners.

Now, using a sub-tree rather than a single element allowed to limit the impact of unexpected behaviour and unexpected variability. First, as mentioned, the first part of the algorithm returns element identifiers, for the Puppeteer API to be able to identify the correct elements. Only, some elements do not have a class name. In that case, the value returned (null) does not allow Puppeteer to identify the element and assess its visibility. Suppose more than one element is being returned. In that case, it reduces the chances that all elements returned do not have a class name and, therefore, increases the chances that Puppeteer will be able to assess the visibility of at least one element. Later, the id of the element was also returned as a backup option if the class name was undefined. This also helped increase Puppeteer’s ability to assess the visibility of the banner elements. Second, if, as is the case quite often, the first part of the algorithm returns more than one element with a valid class name, then the visibility of all those elements can be assessed and the majority decides the classification. This makes the algorithm more resilient to the unexpectedness and variability of websites. For example, it is possible that certain elements within the cookie banner are hidden but that overall the banner and most of its elements are visible. Using the majority to make the decision is an attempt to make sure the algorithm is as close to the true value as possible.

3.6.4 Third, and Final, Iteration Batch

The third iteration of the algorithm differs from the second one in a few main aspects: it no longer starts its search from terminal nodes, takes measures to reduce banner detection false positives, searches iframes, and takes measures to limit visibility false positives.

First, the algorithm no longer starts building a subtree from a filtered set of terminal nodes. While this measure did reduce false positives, it generated new false negatives because some websites had no terminal nodes with word matches. When considering how to pivot, it

was decided to reduce the number of steps. Rather than adding steps to find and filter terminal nodes, the algorithm would traverse the whole DOM itself and generate sub-trees as it traverses the DOM. Therefore, the new iteration of the algorithm loops through all the elements and starts building a sub-tree from that initial element, going a maximum of five levels down from that initial element. The sub-tree creation stops once the maximum number of nodes is reached. Moreover, it excludes the sub-tree from consideration if it has not reached the end of the DOM after 5 levels. The main difference here can be summarised as using a top-down rather than a bottom-up approach.

Second, two additional measures were taken to reduce the number of false positives. That was especially needed as the first modification removed the terminal nodes anchoring, which brought false positives back up. First, a minimum threshold of two distinct matched words has been set for the algorithm to consider the sub-tree as a cookie banner. Therefore, if the best sub-tree candidate only has 1 distinct word matched with the corpus, it is considered as a negative. Similarly, the sub-tree must have at least two different elements as part of it, which was also meant to reduce the chances of an isolated element winning over the banner. Second, the corpus was adjusted. The corpus with the best performance remained the medium corpus - no cookies, but certain terms consistently led to false positives (they were also useful in detecting real banners, but those 2-3 words were responsible for the majority of false positives. Therefore, it was decided that they were not actually needed). The words “privacy policy”, “learn more”, and “preferences” were removed from the corpus.

Third, a key addition was made to this algorithm’s iteration: searching for cookie banners belonging to a different origin. Now that the accuracy results improved, it was time to deal with this special case. Recall the same origin policy and the recursive visit of frames to gather storage data: cookie banners are similar. The HTML of some cookie banners is included inside an iframe loaded from a different origin. Therefore, one must repeat the banner detection algorithm within every frame's context to ensure all relevant HTML is analysed. The recursive search is interrupted when a candidate has been found to ensure this process does not take too long. This is possible because the algorithm's accuracy was high enough that if a candidate has been found, it is likely the correct one.

Fourth, the elements in the subtree were too diluted, leading to false positives in visibility assessments. This is due to the sub-tree being considered including elements outside of the cookie banner, that are visible on the page. It also indicated that the children's check was insufficient to remove this issue. A simple fix for this issue was found: only keep elements with at least one-word match into the final subtree. This means that the subtree is getting reduced in size quite significantly while keeping a certain number of elements on which to check the visibility. This seemed like a good balance, and the accuracy results significantly increased after this switch.

3.6.5 Performance on Top-250 Websites

Appendix 2 provides a visual representation of all the steps included in the final version of the algorithm. The final parameters of the banner detection algorithm are as follows:

- A maximum of 5 levels are considered in any sub-tree
- A sub-tree cannot have more than 15 children
- The corpus used is composed of the 17 following terms: 'agree', 'accept', 'accept all', 'accept cookies', 'consent', 'reject', 'reject all', 'decline', 'cookie preferences', 'manage cookies', 'more information', 'privacy statement', 'cookie policy', 'cookie notice', 'use cookie', 'use cookies', 'uses cookies'
- Elements with no word matches are removed from the sub-tree
- At a minimum, two distinct matches must have been found
- At a minimum, two different elements with word matches must have been found

When it comes to testing the algorithm's accuracy in detecting the presence of a cookie banner and judging the visibility of a cookie banner, an important distinction needs to be made. While the accuracy of the *visibility* of a cookie banner is assessed for every browser, that is not the case for detecting the *presence* of the banner. This is not at all a problem but does require some explaining.

Browsers with cookie banner blocking capabilities do so using a multitude of techniques. Some of these techniques directly impact the HTML of a webpage, where the HTML of the cookie banner is no longer present. For example, this happens when a browser blocks the embedding of an iframe from a different origin, which would have contained the cookie banner. This means that to assess how well the algorithm detects the presence of cookie banners, there would need to be ground truth value about the presence of the banner in the HTML of the websites when visited by different browsers. Since this was a considerable task to do manually, it has not been done. This means the accuracy values for the banner detection are only available for Google Chrome (which does not change the HTML of the page).

This being said it is not because this part of the algorithm can only be assessed for one browser that it is less important. While it cannot be observed for the other browsers with the manually collected data, it is essential that the algorithm be able to detect the presence of a banner when it is present. Indeed, false positives in cookie banner detection will inevitably hurt the accuracy of the visibility assessment, as the algorithm will assess the visibility of non-banner elements. False positives directly impact the accuracy of the visibility detection. Conversely, false negatives wrongfully represent the abundance of cookie banners from a browser's point of view and will also reduce the number of websites considered for the visibility test.

The cookie banner detection accuracy is 87%, with 15 false negatives and 2 false positives out of a sample size of 132 values for Google Chrome. This algorithm does not yield perfect results in detecting the presence of a cookie banner but was judged to have high enough accuracy. Despite multiple adjustments, the algorithm sometimes does not detect a cookie banner on a webpage despite being present in the HTML, within the visited frames, having keywords, etc. The reason for this unexpected behaviour has not been fully understood across many tests being run. Unfortunately, due to time constraints, it was decided that an 85-90% accuracy was enough for the analysis. A possibility is that the cookie banner HTML hasn't loaded by the time the crawler does the evaluation. This had been noticed to have a significant impact and, therefore, a 1-second delay has already been added to the code, which increased the accuracy by around 12%. Between 1 and 5 seconds, no large accuracy jumps were witnessed on the top-250 banners, but it is possible that this pause will not be enough for every website and, therefore, impact the results. Note also that a larger pause would have damaged the crawl pace.

Table 1 shows the accuracy of the algorithm's visibility assessment, per browser. The accuracy of the visibility is high enough to be satisfactory in the context of this measurement. Indeed, these accuracy results enable comfortable cross-browser value comparisons. In fact, the larger proportion of false negatives than false positives for Google Chrome is even a good thing in this regard (the best would of course be if both were zero). Since Google Chrome will be considered the ground truth for the crawl, a larger proportion of false negatives means the result of the comparison across browsers will err on the conservative side. An underestimation of the other browser's capabilities is preferable to an overestimation, especially if it is known that results are probably underestimating the real values.

Two possible behaviours that could limit the algorithm's accuracy during the crawl have been identified. First is if the tool hasn't had the time to hide the cookie banner. This is

Table 1 Visibility Assessment Accuracy, Per Browser

	Accuracy	False Positive	False Negative	Could not assess visibility (excluded from sample size)
Brave (n = 150)	95 %	4	3	1
Firefox (n = 102)	87 %	1	12	2
Ghostery (n = 146)	95 %	3	5	0
Google Chrome (n = 129)	84 %	2	19	3

especially true for Ghostery, where it is easy to see it takes a second or so for the tool to hide the banner. Users can sometimes even see the tool interact with the banner. The 1-second pause added to let the HTML load might help reduce the impact of this limitation, but it is possible it is still happening (after all, if the HTML loads at the end of the pause, the tools will not have had extra time to work). A second possible limitation, although it hasn't been observed during testing, is that the algorithm selects elements that are part of a hidden layer of the cookie banner. For example, the menu that appears when clicking the "more options" or "learn more" buttons. This menu is hidden until it has been clicked on, but the cookie banner itself is still visible. Such a mix-up could be possible as this menu will use similar words to the real cookie banner. If the algorithm selects enough elements of this hidden layer, it is possible it will win the majority and be declared as hidden which could lead to false negatives. Only, as mentioned previously, this issue was not observed during testing.

4 Results

Overall, two complete UK crawls and a US crawl were made. The two UK crawls visited 10,000 websites from the shuffled list explained in Section 3.2. The US crawl uses the same list but only visited the top 5,000 websites due to time constraints. Due to those same constraints, the US crawl did not crawl from the Firefox browser, as this would have doubled the time required to complete the crawl.

The reason two UK crawls had to be performed is that the first crawl attempted to detect cookie banners in an offline manner, using downloaded HTML. This was due to some misleading manual testing that seemed to indicate this technique could work. Anyhow, the second crawl used the cookie banner detection algorithm described in Section 3.5. The need to perform a second crawl was also an opportunity to tweak some of the crawl parameters to improve performance and resilience (more on that in Section 5.2).

The following sections will analyse the results of the second UK crawl. Section 4.6 will shortly expand on the similitude and differences of the results gathered in the two UK crawls to give the reader a better idea of which results were reproduced and which results varied.

4.1 UK Crawl Results (10,000 Websites)

Table 2 below shows the details of the data collected during the UK crawl. The first column indicates which websites have successfully loaded from Puppeteer's perspective. This means the load event was fired and the crawler attempted to take measurements. Recall that a second try to load a website is made for every website that times out before successfully loading. The second column shows the number of websites for which cookie banner data was collected and added to the database. The third column, which is slightly different from others, shows the number of websites for which storage data has been collected and added to the database. The nuance in this column is that if there is no storage data on a webpage, then no entry will be added to the database. Therefore, the numbers in this column show the number of websites 1) that have storage data and 2) that the crawler was able to collect. Unfortunately, the way the data was being gathered and stored does not allow for an easy way to distinguish websites with no storage data from websites where there was an issue collecting storage data. The fourth column shows the number of websites for which requests have been added to the database. The fifth column shows the number of websites for which response data has been collected. Note that this number is even higher than the number of successfully loaded websites. That is because this data is collected differently than other data.

Table 2 Number of Successful Websites for the UK crawl, per Data Category, per Browser

	Websites Successfully Loaded	**Websites With Cookie Banner Data**	**Websites With Storage Data**	**Websites With Request Data**	**Websites With Response Data**
Brave	7456	7165	4842	7165	7659
Firefox	5401	4411	3942	4408	7657
Ghostery	7354	6957	5505	6957	7665
Google Chrome	6967	6358	5352	6357	7665
Intersection of All Browsers	3817	3869	2521	3865	7627
Intersection of Browsers Except Firefox	5888	5892	3888	5891	7643

The response data is added to the database in real-time, meaning every intercepted response gets added to the database. This means that it does not have to wait until the code reaches the measurement section, which happens after the load event gets fired. That is not an issue because one can easily select only the responses of successfully loaded websites for the analysis (and ignore the values gathered by timed-out attempts). Interestingly, the fact that the response data numbers are all so close together indicates that around 2373 websites got non-timeout errors for every browser.

Below the rows detailing the number for every browser are two significant rows. They describe the number of websites that have been visited by all four of the browsers, or by the three browsers except Firefox (due to its considerably lower load success rate). These rows describe the subsample size used when analysing each of these categories of data. Indeed, when comparing the number of cookie banners detected, for example, it is essential to consider only the subsample of data points common to each of the browsers.

The report will focus on the subsample of websites visited by all four browsers, for each cateogry. Note that most of the analyses presented below have been reproduced using the larger sample size of websites (when Firefox is excluded). No significant change in trend have been found by using this larger sample size.

4.2 Cookie Banner Results

4.2.1 Cookie Banner Abundance

This section analyses the abundance of cookie banners as detected by the algorithm, when visited from the Google Chrome browser. This section aims to help quantify how abundant cookie banners are in the crawled websites. The comparison across browsers will be made in the next section, when assessing the visibility values.

The crawl results indicate that cookie banners were found on 945 websites out of the 6358 websites for which Google Chrome collected cookie banner data. In percentage terms, a cookie banner has been detected on about 14.9% of the websites visited. Depending on the reader's expectation, this result might sound surprisingly low, so more detailed explanations might be required.

First, recall that the accuracy test on the top-250 websites showed that the algorithm had a tendency to have a larger proportion of false negatives. This means the 14.9% might be an underrepresentation of the true value.

Second, an important factor to remember is that the algorithm detects only cookie banners written in English. Only, a certain portion of the websites visited by Google Chrome were not written in English. For example, 5% of the list of 10,000 websites use the .ru (top-level domain of Russia) or the .de (top-level domain of Germany) domains. It is likely that most of these websites are written in Russian or German. The result should, therefore, not be read as "only 15% of websites have a cookie banner" but rather as "at least 15% of websites have a cookie banner". This limitation, and why it is not a big limitation, is described in more detail in Section 5.1.4.

Third, some might still compare this 15% value and be sceptical about it when comparing it to the result of previous research. For example, the results of Degeling et al. (12) show that 62.1% of the 6,579 websites they crawled have cookie banners. The important details to notice is that those websites are all European websites, and the websites they crawled are all very popular (top-500 websites of each country in the EU). Similarly, van Eijk et al. (23) detect cookie banners in 40% of the websites they visit. Those websites are, again, top websites from 16 European countries, with the addition of Canada and the United States. These two factors, the location of the websites being visited and their popularity, likely have a lot to do with the higher number of banners being observed.

Indeed, European websites are much closer to the GDPR and probably have had to change how they operate since that regulation came into place. Van Eijk et al. (23) showed that the top-level domain of a website had more of an impact on its behaviour than the user's vantage point. Therefore, even if websites were visited from a UK vantage point, websites with TLDs far removed from European legislation probably haven't been impacted by GDPR all that much.

Additionally, the UK crawl made as part of this measurement visits websites all across the distribution of popularity between the most popular and the 100,000th most popular. This should already be cause for caution when comparing the results of this crawl with the results of crawls targetting the 100-500 most popular websites. In fact, Table 3 below shows some data put together as an attempt to see if there is a correlation between a website's popularity and the presence of a cookie banner.

Table 3 Banner Detection Results, per Tranches of Popular Websites

	Algorithm Detection	**Number of websites successfully considered post-crawl**	**True Value**
Top 1-250 Websites	44.7%	132	55%
Top 1-1,000 Websites	30%	530	NA
Top 1,000-2,000 Websites[1]	15.6%	772	NA

The picture painted by the table seemed to point to a gradual decline in the presence of cookie banners as one went further down the distribution of website popularity. However, the table is still very focused on a small sample size and on websites that are still very high in the website distribution. That is why Figure 4 is presented below.

It shows the percentage of websites on which cookie banners have been detected over all the websites crawled. The websites have been pooled in tranches of 2,000 based on their index in the Tranco list. The tranche size was chosen to ensure the sample size in each tranche was sufficiently big to not create too much variability while showing the higher values found in the first 5,000 websites. With tranches of 2,000, the average tranche has a sample size of 127.12 websites, with the smallest tranche having only 103 samples. The largest sample is the first tranche, with 559 samples, as the first 1,000 websites are all part of the list.

The first three tranches are three of the four only tranches surpassing the 20% threshold, with an average of 27.4%. On the other hand, 15 of the 17 last tranches (from index 66,000 to 100,000) are below the 10% threshold, with an average of 7.3%. Large variations (e.g. at index 8,000) should be ignored as they are attributable to the small sample size, not to the underlying values.

This graph confirms that there is a downward trend in the percentage of websites for which the algorithm detected a cookie banner. This can be attributed to two main causes. The first one is that the abundance of cookie banners does, in fact, go down for less popular websites. The second cause is that the cookie banner detection algorithm could get worse at detecting cookie banners on less popular websites. That would imply a considerable difference in the wording of cookie banners, as this is the technique used to detect the cookie banners.

[1] Note that a separate crawl was conducted to visit the top 1,000-2,000 websites to test this hypothesis.

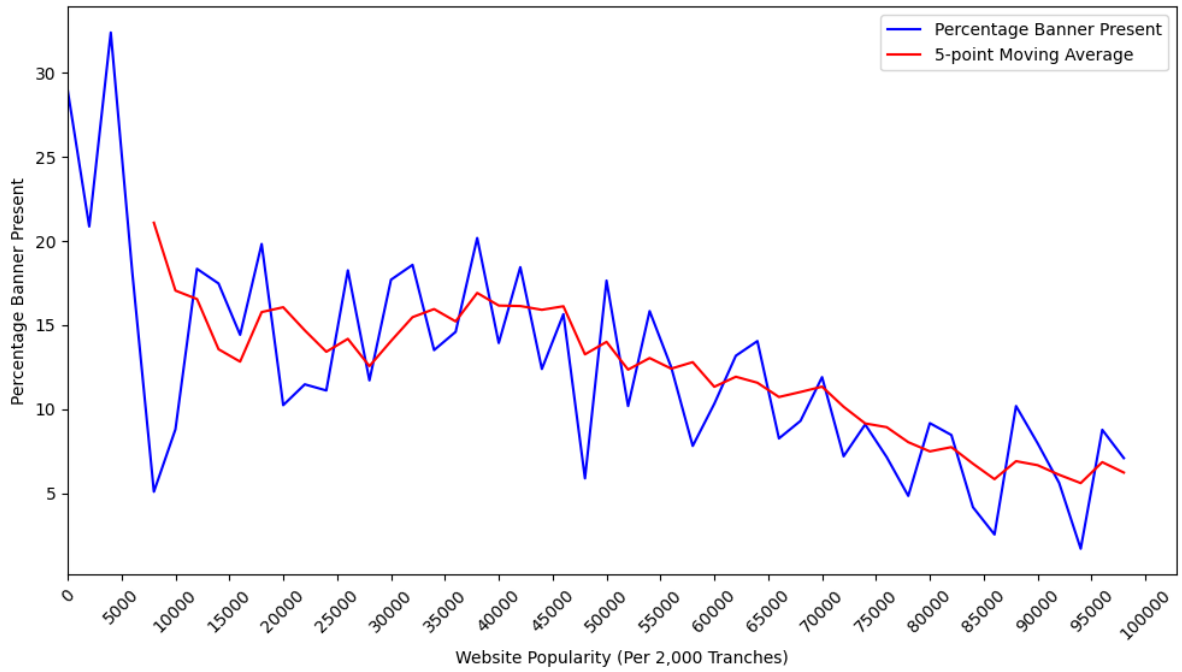


Figure 4 Cookie Banner Presence (Percentage), per Website Popularity Tranche

4.2.2 Cookie Banner Visibility

As mentioned when describing the algorithm, the visibility of cookie banners is assessed after a cookie banner has been detected on a webpage.

Before analysing the visibility results of the subset of websites visited by all browsers, the visibility data of Google Chrome will be analysed. Out of the 945 cookie banners detected, the algorithm judges that 850 of them are visible. In percentage terms, cookie banners are visible on 13.4% of websites visited by Google Chrome. That result indicates that 10% of the detected cookie banners have not been classified as visible. As Google Chrome does not have any cookie banner blocking capability, and that the testing on the top-250 wesbite never flagged a hidden, but present, banner, it is probable that the true value is that every detected banner is visible, and therefore that this 10% of invisible banners fall within some sort of error.

This can be caused by two causes: false negatives assessing the visibility or Puppeteer being unable to assess the visibility of the cookie banner elements. Unfortunately, an error in the code makes it impossible to distinguish between the two cases. This is an unfortunate limitation, which could be fixed if future iterations of the crawl were conducted. The testing data on the top-250 banners indicates that both cases are rarely encountered. Only 4 out of 129 (3%) instances have been tagged as present and invisible. Additionally, the algorithm encountered an issue assessing the visibility of banners on only three websites out of 132[2] (2.3%). Translating these data points from the testing environment to the crawl results, one could expect about 5% of banners detected to be classified as non-visible. Unfortunately, due to the considerable sample size difference, it is difficult to pinpoint exactly why an extra 5% of invisible banners exist. As mentioned in Section 3.6, a possible culprit is that cookie banners are not given enough time to appear on the page before the measurement has been taken. As much preventive measures have been taken to minimise this possibility, but it could be responsible for some of these extra invisible assessments.

[2] The sample size being 132 rather than 129, is due to the fact that in the top-250 tests, banners for which the algorithm could not assess the visibility were exlcluded from the subsample of pages considered for the visibility accuracy test.

Now that the Google Chrome data has been thoroughly analysed, it is time to look at the data that is probably the most central for this project: How effective are the different browsers being studied at hiding cookie banners? (Here, hiding simply means "not visible". A distinction between the techniques used to hide banners is explained in Section 4.2.3)

Figure 5 shows the percentage of websites with visible cookies within the subset of the 3869 websites visited by all browsers for which banner detection data is available. In this subset of websites, Google Chrome has the most visible banners at 390 (10.1%). It is followed by Firefox at 327 (8.45%), Ghostery at 165 (4.3%) and Brave at 51 (1.3%).

Next to it, Figure 6 offers a different perspective of the diminution of cookie banner visibility. It presents the percentage change of each browser compared to Google Chrome (lower is better). Recall that Google Chrome is the proxy for the real number of visible cookie banners, and that this comparison is being made using the same subsample for every browser.

First, the performance of Brave and Ghostery. Both of these privacy-oriented browsers seem to be very effective at hiding cookie banners within 1 second of the page loading. Indeed, Brave displays 86.9% fewer cookie banners than Google Chrome. Similarly, Ghostery displays 57.7% fewer cookie banners.

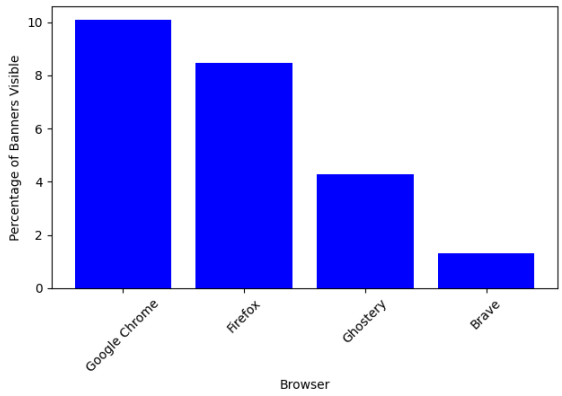


Figure 5 Percentage of Cookie Banner Visible, per Browser

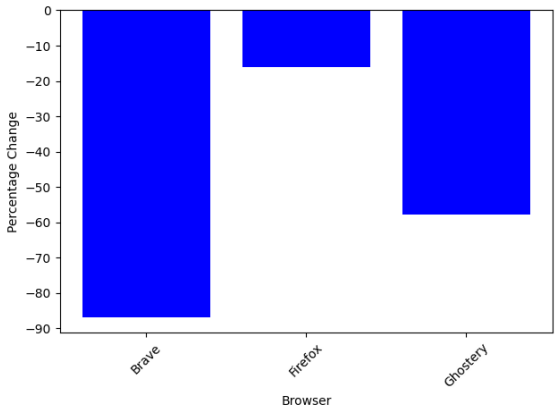


Figure 6 Percentage Change in Cookie Banner Visibility Compared to Google Chrome

Second, the performance of Firefox. Firefox is showing 16.2% fewer cookie banners than Google Chrome. This result seems to indicate that Firefox is less efficient at blocking cookie banners than Ghostery or Brave. While this is certainly a possibility, it must be said that this result is not completely coherent with the numbers found both manually and by the algorithm in the top-250 websites testing environment.

Table 4 below shows the percentage change in cookie banner visibility for the three browsers in the testing environment. Interestingly, it shows both the values predicted by the algorithm as well as the true values observed manually. The Firefox results in that environment indicate that 55.7% (manual) or 57.1% (algorithm) fewer banners are shown compared to Google Chrome. Clearly, those results indicate that Firefox is blocking a larger proportion of cookie banners than the results of the crawl. This is mainly attributable to the fact that the top-250 websites are not representative of the whole distribution of websites. Table 3 and Figure 4 examplify this very well by showing how the abundance of cookie banners in top websites drops drastically for less popular websites.

Table 4 Cookie Banner Visibility, Compared to Google Chrome, Top-250 Websites

	Percentage Change in Cookie Banner Visibility - Manually Observed	Percentage Change in Cookie Banner Visibility - Detected by Algorithm
Firefox	-55.7 % (n = 162)	-57.1 % (n = 86)
Ghostery	-86.9 % (n = 165)	-81.3 % (n = 123)
Brave	-92.9 % (n = 161)	-88.0 % (n = 126)

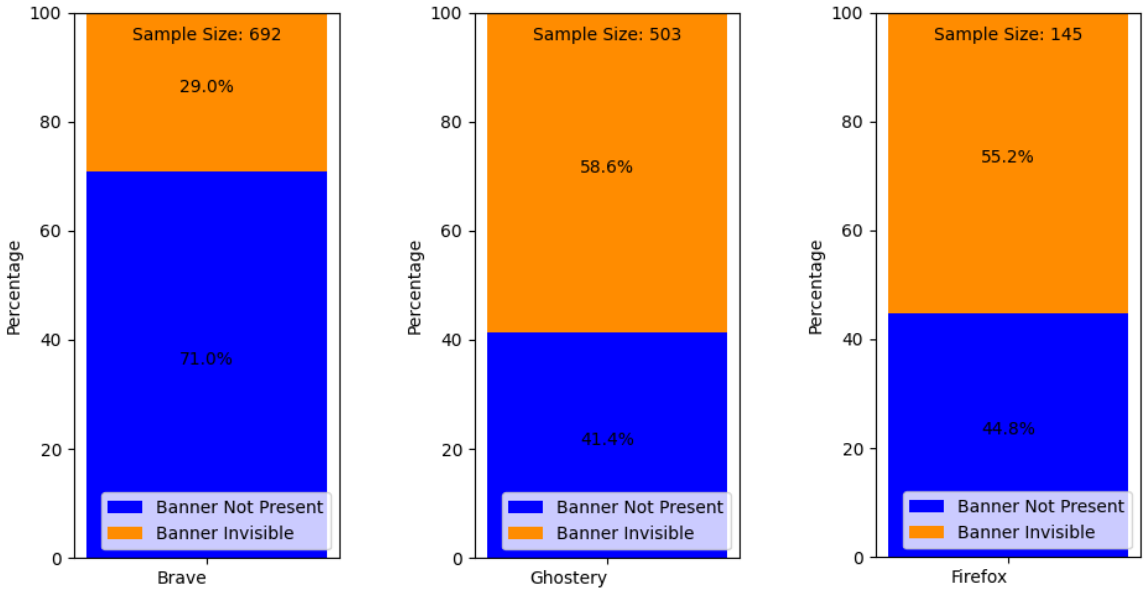
The Ghostery results echo the fact that the top-250 websites are not the best proxy for the internet. Indeed, similarly to Firefox, the percentage change in cookie banner visibility dropped considerably for Ghostery between the test and crawl environments. It went from 81.3-86.9% (algorithm-manual) to 57.7% fewer.

To sum up, the inconsistencies between the results of the top-250 banners and the crawl results seem to imply that the sample of websites considered has an impact on the visibility results. The Brave results are very consistent across all the data available, which indicates that it does maybe, really, block around 85-90% of cookie banners observed on Google Chrome. On the other hand, the Firefox and Ghostery data do not allow to make this statement with the same level of certitude. Nonetheless, two other takeaways can still be made. First, both Ghostery and Firefox are able to block cookie banners that would otherwise be displayed to the user. Second, Firefox's performance is inferior to the one of Ghostery and Brave, and the data seem to indicate that Ghostery's is inferior to Brave's.

4.2.3 Cookie Banner Blocking Techniques

The goal of this section is to analyse *how* browsers block cookie banners. The algorithm can pick up two different states. Either the cookie banner is hidden, meaning it has been detected in the HMTL but is not visible, or it is blocked, meaning the cookie banner is not present in the HTML despite knowing the website normally has a cookie banner. To clarify the latter, websites for which Google Chrome detected a cookie banner are considered to have had a cookie banner in the HTML. Therefore, if the cookie banner is not detected in the HTML of the website when visited by the other browsers, this means it has been blocked.

Figure 7 Proportion of Banner Hidden versus Blocked, per Browser (Different Sample Sizes)



The sample considered in this section varies for every browser. This is due to the intersection of websites where Google Chrome detects a banner, and where the banner is not visible for any of the three browsers, being too small. Therefore, the sample considered for every browser are websites for which Google Chrome detects a banner, and where a banner was not visible from the other browser's perspective. The subsample of each browser therefore includes all the websites where the cookie banner blocking behaviour is being expressed.

Overall, as Figure 7 shows, Ghostery and Firefox seem to be behaving very similarly. There is only a 3.4 basis point difference between the proportions, which represents a 6.2% change. On the other hand, Brave behaves differently from both Ghsotery and Firefox. Comparing the Ghostery and Brave values shows that the algorithm goes from not detecting a banner on 41.4% of websites with Ghostery to 71.0% for Brave. This is a 29.6 basis points increase, which represents a 71.5% increase from the Ghostery value. This means that Brave is about 30% more likely to be blocking a cookie banner, rather than hiding it, compared Ghostery and Firefox.

As was mentioned in Section 3.6.5, no tests assessing the accuracy of the algorithm at detecting cookie banners for the Brave, Ghostery, or Firefox browsers have been made. This is because of the complications involved in collecting the ground truth values, which would require manually inspecting the HTML of every website not showing a cookie banner to see if it has simply been hidden or removed. However, there is no reason why the algorithm should be worse at detecting banners for these browsers than Google Chrome. Therefore it can be safely assumed that the accuracy of the algorithm of this task for these browsers lies around the Google Chrome accuracy of 85%.

For the hidden banners, the analysis does not differentiate between a decision having been registered, thus hiding the cookie banner, or the banner having been hidden without any interactions having been made with it. For blocked banners, the current analysis does not differentiate between the banner never being embedded in the HTML, or the banner being removed before the crawler could make the measurement.

The different techniques all have slightly different implications. Registering a decision on the cookie banner, such as a non-consent should normally ensure no tracking is being done. Only, as Section 2.3.1 exposed, there are reasons to doubt this would actually happen. As an alternative, simply hiding the banner could be a good alternative as it does not rely on the non-consent being respected, and it does not actively interact with consent-tracking systems (though they are still present on the page). Finally, proponents of removing the cookie banner altogether argue that this is the most privacy-preserving option as it has all the pros of hiding the banner, in addition to removing (or blocking the installation of) the consent management code, hence adding to the distance between the user and those systems. Moreover, if the blocking is made at the request level, this could completely cut off third-party consent managers and be even greater for privacy as no information, not even the user's IP address, would be communicated to them.

The impact of the technique used to block the cookie banner, as talked about in this section, is not fully isolated in the following analyses which look at browser storage and at requests. Only, while this attribute isn't fully isolated, recall that each browser is as close to its default settings as possible, and that the crawler does not interact with any of the websites it visits. Therefore, all variation in results should be attributable to the differences in the privacy-preserving features of the browsers (of which the technique used to block cookie banners is a part of).

4.3 Browser Storage Analysis (Cookies and Local Storage)

This section analyses the data related to browser storage, meaning cookies and local storage. The subset of websites being considered here comprises the 2,521 websites for which data is available for every browser.

Recall that the difference between the subset of loaded sites and of sites with storage data does not mean that an error collecting storage data occurred every time. Instead, because of the way the data is collected, websites with no storage value do not figure in this subset. This is unfortunate as it makes it a lot harder to analyse the proportion of websites with no storage for each browser. This mechanism could be modified in a future data-gathering effort in order to make it easier to perform this type of analysis.

Figure 8 shows the total number of cookies and local storage for every browser in this subset of 2,521 websites. It is clear that Google Chrome has a larger total number of storage units. On average, this represents 39.0 storage units per website in the subset for Google Chrome, 31.8 for Firefox, 12.8 for Ghostery, and 8.4 for Brave. Recall that the average across all websites visited would be a lower value as websites with zero browser storage would bring down the average.

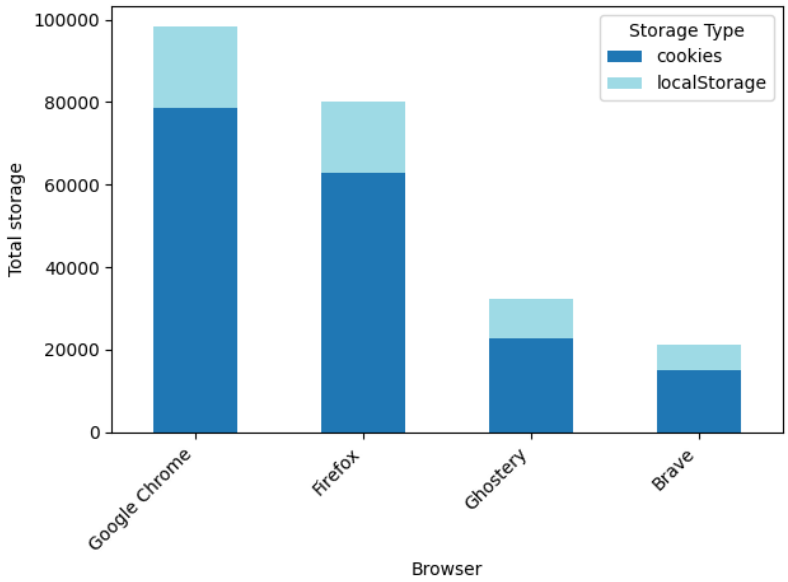


Figure 8 Total browser storage, per Browser

It is also easily observable that cookies represent a larger proportion of storage than local storage, for every browser. As a reminder, browser storage, such as cookies and local storage, can be used to store trackers by storing values that identify a user in the browser storage. Therefore, limiting the number of cookies and local storage being downloaded is a good thing from a privacy perspective.

Below, Figure 9 represents the reduction percentage in cookies and local storage for each browser, compared to Google Chrome (lower is better). The ordering of the browsers, from largest to smallest reduction, is the same as when looking at the cookie banners. Brave is the browser with the largest decline in storage, at 78.5% overall. It is followed by Ghostery at 67.1% and Firefox at 18.4%.

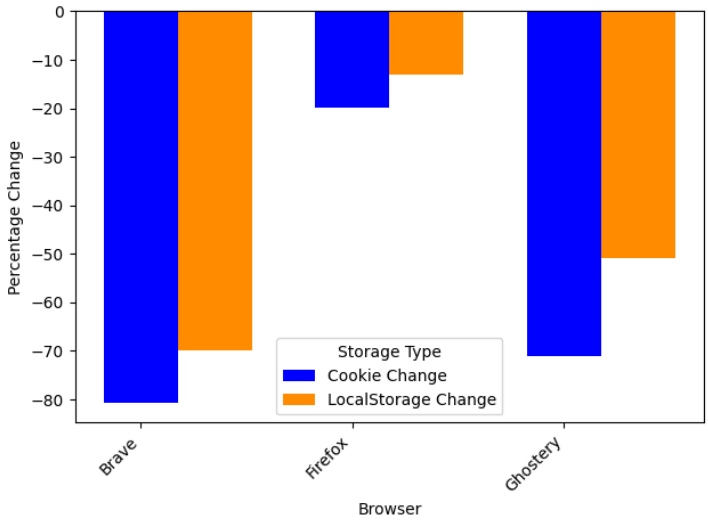


Figure 9 Percentage Change in Storage Compared to Google Chrome, per Browser

As mentioned multiple times, this work does not distinguish between essential and tracking cookies, nor does it count the data on sites storing no values before consent. In fact, no user consent is ever given during the crawl. Google Chrome is not expected to ever interact with cookie banners, Brave is expected to hide the banner without interacting with it, Ghostery is expected to select the most privacy-preserving option when possible or hide the banner, and Firefox is expected to reject cookies or leave the banner visibility. One this is common across these scenarios: no consent is ever given. Therefore, according to GDPR, only essential storage should be used but no trackers should be install before registering consent. Despite this, the amount of storage found in the browsers varies dramatically. This significant variation indicates that more than just essential storage is being installed. While this study does not try to deduce how many of the 21.1k Brave storage values (8.4 per website, on average) are essential versus tracking, it seems very likely that as part of the extra 77.2k storage values (extra 30.6 per website, on average) being installed on Google Chrome are trackers being installed. Again, these trackers are either being install before any consent is given (for the techniques not interacting with the cookie banners), or they are being installed despite registering a non-consent on the cookie banner (for those interacting with the banners). In either case, this finding echoes Trevisan et al.'s (17) result that non-essential cookies are used before user consent is given on about half of the websites they studied. This also shows the need for the other privacy-preserving features (blocking trackers for example), as trackers seem to be in use even when the user does not consent to their use.

4.3.1 Third-Party Browser Storage

Third-party storage is identified when the frame origin associated with the storage differs from the website URL (the top-level frame). Table 5 shows the subset of websites that have been detected to have third-party storage within the sample of 2,521 websites with storage data for the three browsers. As the table indicates, Brave is blocking third-party browser storage on 24% of websites, Ghostery on 18% and Firefox on about 5%.

Table 5 Number of Websites With Third-Party Storage

	Number of Website With Third-Party Storage	% of websites from the sample using third-party storage (from 2,521)	Variation versus Google Chrome
Brave	1118	44.3%	-24.0%
Firefox	1404	55.7%	-4.6%
Ghostery	1205	47.8%	-18.1 %
Google Chrome	1472	58.4%	NA

Table 6, exposes a different perspective. For each browser, it pools websites in categories after assessing whether the website holds more, equal, or less third-party storage units. This helps understand how many websites have a reduced number of third-party storage, broken down by studied browser. The results indicate that 80.5% of websites visited by Brave have fewer third-party storage values when compared to Google Chrome. That number is 70.7% for Ghostery and 47.9% for Firefox. This shows very clearly that each of those browsers, to varying degree, reduce the number of third-party storage on websites.

Another interesting measure is the proportion of websites on which all third-party storage has been removed. This is the case on 25.0% of websites visited by Brave, 19.5% by Ghostery and 9.5% by Firefox. There are however some websites (3%, 10% and 25.8% for Brave, Ghostery, and Firefox respectively) which have more third-party storage than Google Chrome. The value for Brave is quite low, which could mean it falls within measurement errors, but the other two values seem to indicate that some websites do manage to have extra third-party storage compared to Google Chrome. Those values are still considerably lower than the portion of website on which third-party storage has been reduced or eliminated.

Table 6 Classification of Websites, per Browser, Comparing The Number of Third-Party Storage Units to the Google Chrome Value

	Category (Comparing Third-Party Storage)	Number of Websites	% of websites from Google Chrome's third-party subsample (n=1472)	% of websites from Google Chrome's third-party subsample (n=1472)
Brave	Less (zero)	368	25.0%	**80.5%**
	Less (non-zero)	819	55.6%	
	Equal	243	16.5%	16.5%
	More	44	3.0%	3.0%
Firefox	Less (zero)	140	9.5%	**47.9%**
	Less (non-zero)	566	38.4%	
	Equal	387	26.3%	26.3%
	More	381	25.8%	25.8%
Ghostery	Less (zero)	287	19.5%	**70.7%**
	Less (non-zero)	755	51.2%	
	Equal	284	19.3%	19.3%
	More	148	10.0%	10.0%

As for a view of the total number of third-party storage units being installed on the different browsers, Figures 10 and 11 below show the details (lower is better for Figure 11). The figures might seem very familiar, that is because the trend for third-party storage is very

similar to the one detected in the previous section when analysing all the storage values. The proportion of total storage that is third-party storage is about 50% for both Ghostery and Firefox. It is at its highest with Google Chrome at 56.7%, and at its lowest with Brave at 41.7%. This reduction in third-party storage is coherant with the breakdown from Table 6. When pooling the reduction in third-party cookies and local storage (Figure 11), Brave has a 84.2% reduction compared to Google Chrome, Ghostery a 71.2% reduction and Firefox a 28.5% reduction.

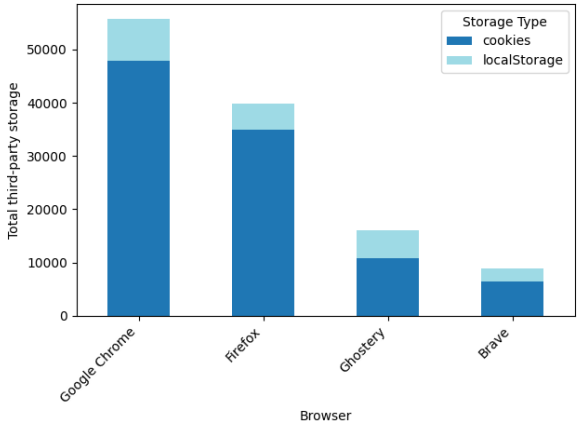


Figure 10 Total Third-Party Storage, per Browser

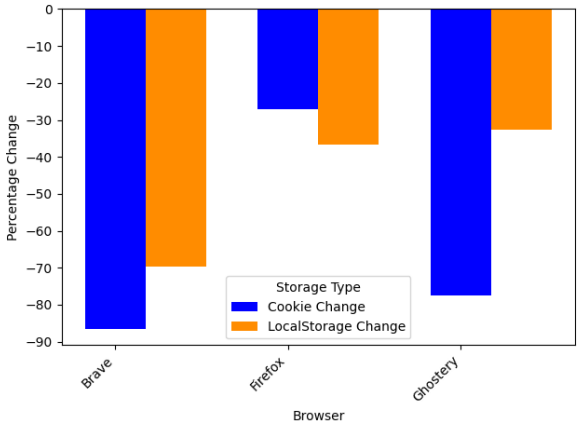


Figure 11 Percentage Change in Third-Party Storage, Compared to Google Chrome, per Browser

4.4 Requests

4.4.1 Total Number of Requests

The number of requests made by a website are also important to analyse, as they can also impact user privacy. UID smuggling, which was described briefly in the background section, is an example of a mechanism that uses requests to exfiltrate user data. Requests can also be used to download a multitude of trackers onto a webpage (e.g., cookies, invisible pixels, scripts, etc.). Therefore, as a general rule of thumb, limiting the number of non-essential requests is best for user privacy.

Below, Figure 12 provides information on the total number of requests made, and Figure 13 shows the percentage change (lower is better) when comparing browsers to Google Chrome, using a subsample of 3,865 websites. The order in which the browsers are found is the as in the cookie banners and total storage analyses. Google Chrome is the browser with the most requests, with an average of 61.9 requests made per website. It is followed by Firefox with an average of 61.2, Ghostery with 55.7 and Brave with 52.6. This represents a 1.1% reduction of overall requests for Firefox, a 10.0% reduction for Ghostery, and a 15.0% reduction for Brave. Based on the number of requests being made, Brave performs better than the other browsers.

As Figure 12 shows, the proportion of third-party requests is very stable across browsers, and represents on average 37.7% of the requests made. Given the abundance of both types of requests, it is interesting to look at the variation of the number of requests, per party type (Figure 13). Firefox probably has the most interesting result, due to its 4.5% increase in the number of third-party requests compared to Google Chrome. Otherwise, both Brave and Ghostery witness a reduced number of requests from both first and third parties, with the first party decrease being larger.

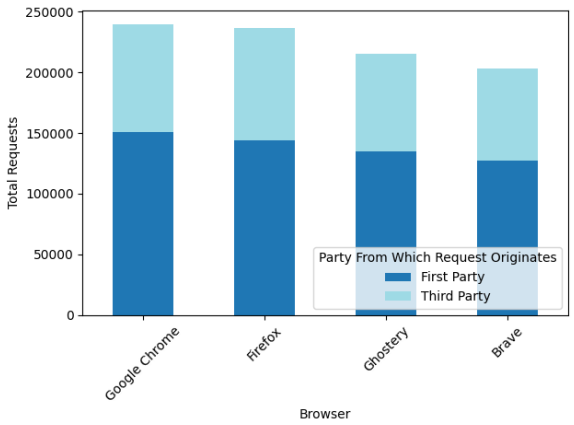


Figure 12 Total Number of Requests, per Party Type, per Browser

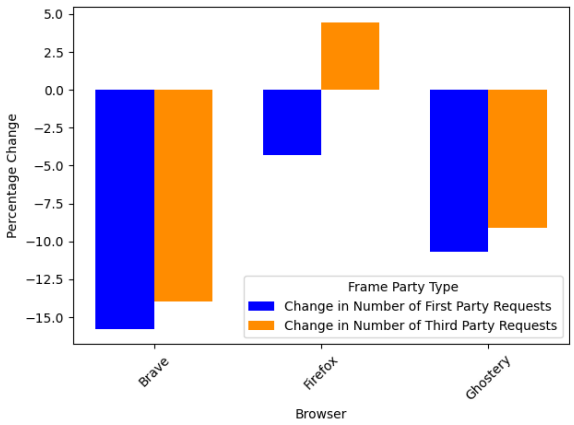


Figure 13 Percentage Change of the Number of Requests, per Party Type, Compared to Google Chrome

4.4.2 Frames Making Requests

In addition to analysing the total number of requests being made, it is also relevant to analyse who makes those requests. This section analyses the number of distinct frames making requests and breaks them down into first-party and third-party frames. Similarly to the storage analysis, third-party frames are those having a frame origin different from the page URL (top-level frame).

Table 7 Number of Frames, per Party Type, Making Requests

	First Party		Third Party	
	Number of Frames	**Number of Websites**	**Number of Frames**	**Number of Websites**
Brave	2545	2545	1459	1653
Firefox	2575	2575	1813	1932
Ghostery	2540	2540	1471	1742
Google Chrome	2544	2544	1816	2057

As Table 7 above, and Figure 14 below both show, there is little to no variation in the number of first-party frames. This is not entirely surprising, given that first-party frames are likely to make essential requests to render the content of the webpage, etc. On the other hand, third-party frame data tell a different story. First, looking at the number of different frames and at the variation shown in Figure 14, Brave and Ghostery register about 19.5% fewer distinct third-party frames than Google Chrome (and Firefox, which has a similar number of third-party frames). Having fewer third-party frames making requests is a positive result, as it limits the number of third parties which could exfiltrate tracking information.

The number of websites for which third-party requests have been registered also points to interesting browser behaviour. Table 7 clearly shows that the number of websites for which third-party requests have been registered varies between browsers. Taking Google Chrome as the control value, it is observable that every other browser entirely eliminates third-party requests from a certain number of websites. Indeed, 19.6% fewer websites visited by Brave, than by Google Chrome, register any third-party requests (Figure 15). That value is 15.3% for Ghostery and 6.1% for Firefox. Obviously, eliminating all third-party requests made from websites is very beneficial from a privacy perspective.

Finally, one last interesting insight can be drawn from Table 7. Notice that the number of distinct third-party frame origins is always smaller than the number of websites in the sample

size. This is an indication that some third-party frames are being detected on more than one website, and this for every browser. Further analysis could be made to see which frames are most often encountered, and on what proportion of websites visited they are found, but this is not central to this analysis.

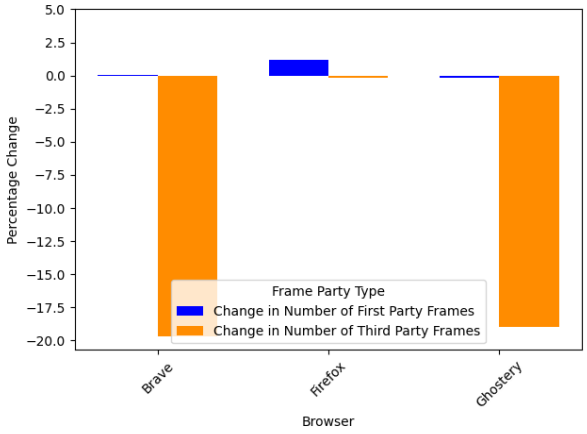


Figure 14 Percentage Change in Number of Frames, per Party Type, per Browser

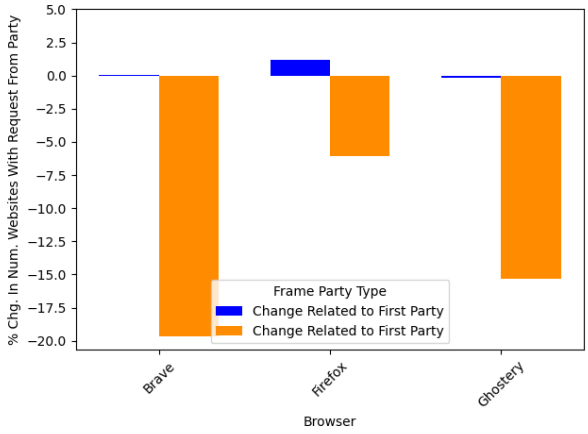


Figure 15 Percentage Change in Number of Websites Registering Requests of a Certain Kind, per Browser

4.5 Responses

4.5.1 Total Number of Responses

The response data (Figure 16, 17) resembles the trend found with request data, which is unsurprising as requests and responses are linked. The average number of responses received is highest with Google Chrome at 60.2 per website and minimized with Brave at 49.7 per website. Ghostery and Brave see a 15.6% and 17.4% decrease, respectively, compared to Google Chrome, while Firefox is pretty much at par.

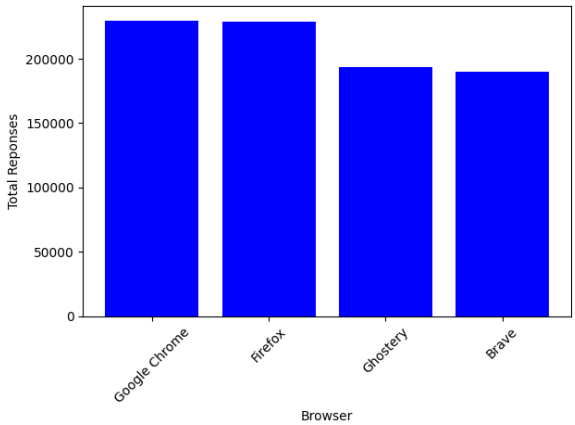


Figure 16 Total Number of Responses, per Browser

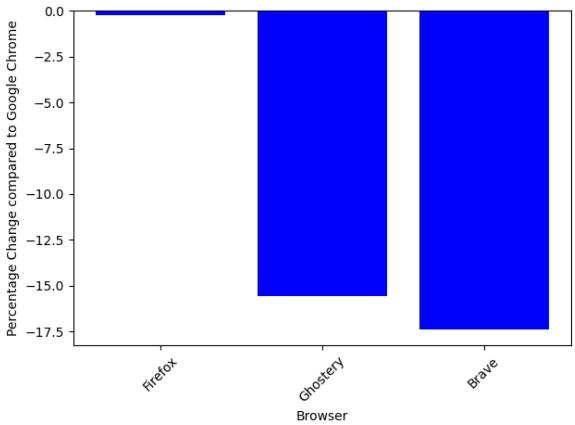


Figure 17 Percentage Change of Total Responses per Browser compared to Google Chrome

4.5.2 Content-Types

No discernable change in the different content types (ignoring the charsets) being let through by the browsers. The number of distinct content types being observed at least once varies very little (Brave: 195, Firefox: 186, Ghostery: 171, Google Chrome: 217). The list of extra-content types being observed by Google Chrome does not seem to indicate any pattern,

other than internet variability. Indeed, most of them are observed between 1 and 5 times, almost nothing compared to the total number of responses.

As for the frequency of the content types, the proportion and order of the top-10 content types do not offer any clear insight. Neither does the percentage change in abundance for selected content types compared to their abundance in Google Chrome.

4.6 Location Impact: Comparing UK and US Results

As a quick reminder, the US crawl consisted of the first 5,000 websites of the UK crawl and Firefox was not part of that crawl. Table 8 below shows the breakdown of the number of websites per data category being collected.

Table 8 Number of Successful Websites for the US crawl, per Data Category, per Browser

	Websites Successfully Loaded	Websites With Cookie Banner Data	Websites With Storage Data	Websites With Request Data	Websites With Response Data
Brave	3743	3674	2569	3674	3820
Ghostery	3781	3667	3095	3667	3821
Google Chrome	3463	3461	3104	3461	3818
Intersection of Browsers	3356	3357	2444	3357	3812

4.6.1 Cookie Banners Visibility

A subset of 1643 websites has been created. It consists of the websites for which there are cookie banner data for three browsers in both of the crawls. The goal of this subset is to compare the behaviours of websites when visited from the UK and the US.

Table 9 Cookie Banner Visibility Comparison, UK vs US

	Number of Visible Banners - UK	Number of Visible Banner - US	% change (from UK to US)
Google Chrome	176	119	-32.4 %
Ghostery	67	45	-32.8 %
Brave	19	19	0 %

The Google Chrome values indicate that cookie banners are visible on 10.7% of the websites in that sample from the UK vantage point. That value drops to 7.2% from the US vantage point, which, as the table indicates, represents a 32.4% drop. As for the detection of banners (rather than the visibility), cookie banners are detected on 11.9% of websites from the UK vantage point and 8.6% of websites from the US vantage point. This represents a 27.7% drop in cookie banner presence, which is coherent with the cookie banner visibility variation found above. Overall, it seems that a user visiting those websites from the US would see around 30% fewer cookie banners than a similar user in the UK.

4.6.2 Browser Storage

Similarly to the previous analysis, a subset of websites that have storage data for the three browsers in both crawls has been created. The subset contains 1015 websites.

Table 10 Total Storage Comparison, UK vs US

	Total Storage - UK	Total Storage - US	% change (from UK to US)
Google Chrome	34,730	53,865	55.1 %
Ghostery	12,788	16,512	29.1 %
Brave	8,843	9,041	2.2 %

The 55.1% increase for Google Chrome represents, on average, an extra 18.9 storage values per website in the subset. This is a clear indication that extra trackers are being installed on the browser of the US crawler. Similarly to Section 4.3, there is no need to know exactly which proportion of this storage constitutes trackers versus essential storage. The scale of the change exceeds what could be attributed to essential storage. Especially when taking into account that here, as opposed to Section 4.3, there is no possible cross-browser variation as Google Chrome is compared with itself.

The 29.1% increase for Ghostery indicates that it is able to block a good proportion of the added storage. Indeed, only 3.7k extra storage values were added, compared to Google Chrome's 19.1k. This is, on average, a reduction 15.2 storage values per website.

Brave has by far the most impressive performance. It barely sees an increase in the amount of total storage being installed. Assuming all the extra storage units tried to be added to the Brave browser, Brave filtered out 99.0% of them.

Table 11, offers a similar viewpoint but focuses only on the storage associated with third parties (different frame origin than website URL). The table shows that third-party storage increased by 68.1% for Google Chrome. Knowing that the rate of first and third-party storage combined increased by 55.1%, it is clear that third-party storage grew by a larger proportion than first-party storage. Third-party storage often has tracking abilities, reinforcing the point made before that additional trackers are being installed in Google Chrome when visiting websites from the US.

Brave and Ghostery both perform similarly. They both block a slightly larger amount of third-party storage than they did first-party, thus reducing the percentage change from one vantage point to another by 1.8 and 1.5 basis points respectively.

Table 11 Third-Party Storage Comparison, UK vs US

	Third-Party Storage - UK	Third-Party Storage - US	% change (from UK to US)
Google Chrome	19,870	33,407	68.1 %
Ghostery	6,105	7,788	27.6%
Brave	3,641	3,655	0.4 %

As for the proportion of the total storage attributed to cookies or local storage, that value remained constant. In this subsample, the breakdown for the UK is 77% cookies and 23% local storage. For the US it is 75% and 25% respectively.

5 Evaluation

5.1 Limitations

5.1.1 Set Up and Vantage Point

Initially, the goal was to perform the crawl from a residential IP address in the UK and the US. This was to be achieved through the use of proxy servers, which would have been used to change the IP address of the crawler. The proxy servers would have to be residential machines and, therefore, be considered normal residential machines accessing the internet.

To do so, the web crawler was initially set up to run from a Department of Computing (DoC) Linux machine (Figure 18 represents this set-up). All of the browsers being used have been set up so that their traffic is directed to 127.0.0.1:8080 (a puppeteer flag was used for Chrome and Brave, while a browser-wide setting was set for Firefox and Ghostery). A SOCKS5 SSH tunnel was established between that port and the proxy server used for the crawl. Internet queries would then be performed through the proxy servers, and all the information received by the proxy is then sent back to the machine running the web crawler. Finally, all the collected information was stored on a PostgreSQL server hosted on a DoC virtual machine. The virtual machine was used to facilitate the set-up due to user restrictions on DoC lab machines.

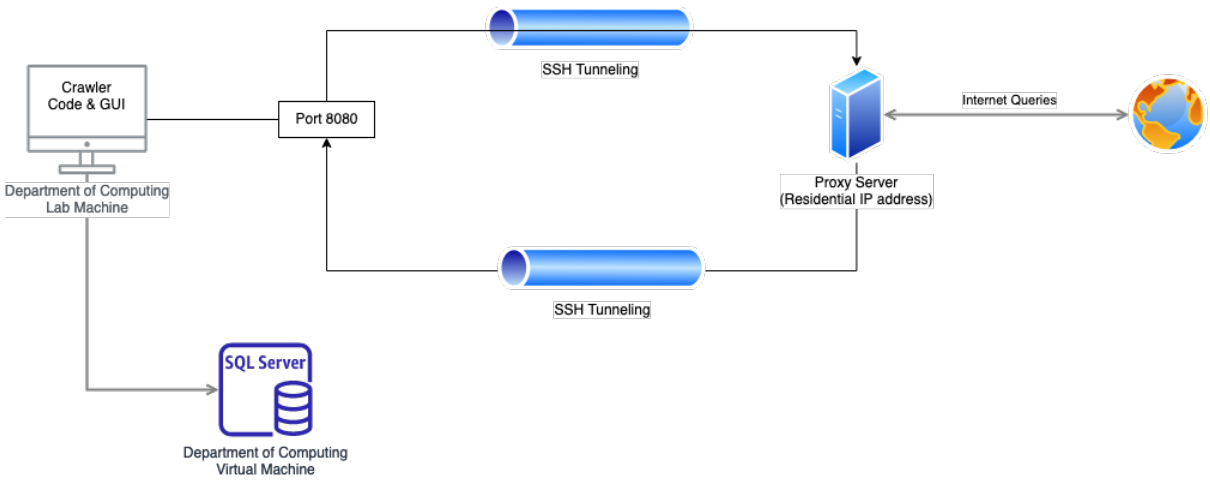


Figure 18 Initial Set-Up Visualisation

Unfortunately, for a few reasons, the set-up was forced to change right before launching the crawler. First, a few days before the crawl launch, the residential proxy server used for the UK residential address crashed. Unfortunately, the server could not be put back online for a few weeks, which meant an alternative was needed. Second, finding a residential server to use for crawling in the US or Canada was more complicated than expected. These two factors led to a change in set-up.

The first alternative was to use a DoC lab machine, which would be connected to a VPN for the US crawl, but that was not possible due to user and network restrictions. The second alternative was to switch the whole crawl set-up to the virtual machine used for the database server. Only, that option required a really extensive and lengthy setting-up phase and was not even guaranteed to work. Therefore, it was decided that the MacBookPro set-up described in Section 3.4 was the best option given time constraints and the reduced risk associated with this option.

The MacBookPro set-up was not initially chosen for two reasons. First, it required buying some equipment (a charger), which was easily avoidable by using the DoC computers.

Second, the internet connection it crawled from can be very weak during peak internet utilisation at the student accommodation. In the end, given the circumstances, these constraints were overlooked. The one notable upside of this crawl set-up is that the UK crawl was performed from a residential IP address, which limited the downside of the UK proxy crashing.

Finally, one last limitation regarding the chosen set-up is using a VPN to modify the crawler's IP address. This was not initially the chosen solution because websites are more likely to detect VPN IP addresses and to change their behaviour (e.g. by adding bot detection tests the user needs to complete such as CAPTCHA). Indeed, VPN IP addresses are more easily detectable as they often have high internet traffic associated with them (from all the VPN users), and the IP can be linked directly to a VPN vendor. Given the constraints described earlier, it was decided that using a VPN connection for the US crawl was better than abandoning the US crawl altogether. It was also decided to have the UK crawl from a residential address, to remove this limitation from the UK crawl, despite this adding another limitation: an external variation that could affect the comparability of results. Indeed, as the US data is collected through a VPN but not the UK data, any change in website behaviour would be hard to isolate and could affect the results of the comparative analysis.

5.1.2 Bot Detection

Probably the largest single limitation concerning the web crawler is the presence of the webdriver flag for the Firefox browser (as well as the Ghostery-branded browser when used). As previously mentioned, the webdriver flag indicates to websites that a bot is controlling the browser, which can often lead to websites with bot-detection verification in place to change their behaviour (like a CAPTCHA page, or refusing to load the page). Despite the Mozilla documentation (51) clearly stating the webdriver property to be a read-only attribute, attempts were made to remove the webdriver flag, as this was seen as a large enough limitation to warrant spending time trying to find a workaround.

The initial attempt at solving this issue was to test the compatibility of the puppeteer-extra-plugin-stealth library (50) with Firefox. Unfortunately, while this library enables Google Chrome and Brave to pass all public bot tests, it does not work with Firefox as of the date of the experiment.

The following two attempts tried to inject some code when the crawler loads a new page to manually disable or delete the webdriver flag. The tested commands were found by looking at some of the code of the puppeteer-extra-plugin-stealth (47) and some online forums describing the same issue (e.g., stack overflow).

The first of the two attempts tried to use the Puppeteer API to inject the code. Unfortunately, using a simple page.evaluate() was not enough as that would only be executed after the page finished loading, by which time the crawler would have been flagged. Only, that approach seemed promising as it successfully modified the webdriver flag (observed by printing the value before and after the script ran). The direct alternative was to use page.evaluateOnNewDocument(), which is a Puppeteer method that runs the desired script before any other script on the page is run. Unfortunately, this method is not yet supported by the Firefox browser.

The second of these attempts tried running the same scripts, but this time from a Firefox extension. A Firefox extension was therefore created. It allowed to run the script as soon as the document object was created. After multiple attempts at reconciling Puppeteer with web-ext (52) which is recommended to run an unpublished extension in Firefox, it was decided to self-distribute the extension for it to be directly added to a Firefox profile. Once it became possible to test the extension using the web crawler (tests using a normal browser were not useful as the webdriver property was already set to false), multiple various script combinations were tested, but none managed to modify the property. It seemed as if the property could be modified as the prints of the property before and after the script's execution showed a change in the property

value, but by the end of the page load, the property was always set back to true and not once did a script manage to avoid the bot detection tool being used (Antibot (49)). At that point, it was decided to give up these "hacky" attempts at changing this property and accept it as a measurement limitation.

5.1.3 Banner Detection

One limitation of this study is that while it can detect cookie banners without relying on a particular banner format or framework, it only detects cookie banners written in English. Two things are to be said about this. First, this was not deemed a significant issue for this study mainly because of its comparative aim. Indeed, this measurement and its analysis are mainly focused on comparing behaviour across browsers rather than trying to give a data point such as "there are cookie banners on x% of websites on the internet". This meant that as long as the algorithm could only detect cookie banners written in English for all browsers, the main analysis' aims were still attainable. Second, if other research desires to address this limitation, it might be as simple as translating the corpus terms to their equivalent terms in other languages. This was not attempted since it was not central to the desired analysis and the researchers could not verify the quality of translated results.

Moreover, the testing phase, where the algorithm's accuracy was calculated, included only the top-250 websites. As was shown in the analysis, the algorithm's results detect a considerable change in cookie banner presence across the popularity distribution of websites. To solidify the assumption that the algorithm performs well across the distribution and reports accurate results, it could have been beneficial to add websites from across the distribution in the testing phase. The top-250 websites were initially chosen because that was the subset of websites where cookie banners were more likely to be found, therefore maximising the training data for the algorithm for a fixed number of websites. As mentioned earlier, manually visiting 1,000 websites was a considerable enough task, hence why the testing sample wasn't expanded further.

Another limitation is the error that led to the loss of nuance between a present but invisible banner and a banner for which Puppeteer cannot assess the visibility. As previously mentioned, this is not expected to affect a large proportion of results, but it is nonetheless a limitation of this data collection. This limitation could easily be fixed if another round of data collection was executed.

5.2 Successes

Now that the main limitations of this project have been laid out, this section will briefly go over the different sections of the project that were especially successful.

As has been mentioned, two series of crawls have been made (considering only UK crawls). The first complete crawl showed some weaknesses in the crawler code that was not evident when testing with the top-250 websites. The overall impact of these weaknesses was that browser instances would become stuck, thus halting the crawler's progress until human action was taken to rectify the situation. This could happen for multiple reasons.

First of which are new windows popping up when visiting a website. If the window which appeared was a fully functional browser page (with a search bar), the crawler usually continued the crawl from there (although this left a trail of non-closed web pages). Otherwise, if it did not have a search bar, then the crawler became stuck. Second, if a website automatically initiates a download prompt, Puppeteer could not always close the page. Third, and this was the case mostly for Firefox and the Ghostery-branded browser, some webpages seemed to continue loading forever. It seemed that the navigation timeout, or the other timeouts associated with data collection, did not always work or cover every edge case. Fourth, probably due to a combination of factors, it happened more than once that the browser would crash or become

unresponsive (e.g. Application Not Responding in Mac). All of this is to say that the crawl's first iteration required some human interaction to ease the process.

Before the second crawl was launched, adjustments were made to the crawler code to make it more resilient. With these adjustments, the crawl was completed almost without any human intervention (a browser became unresponsive at one point). The added overarching 30-second timeout seems to have covered any edge case not before considered. This ensured that browsers did not remain stuck on a webpage for longer than permitted, which would otherwise have slowed the crawler and impacted the number of requests and responses observed. The new mechanism to close all open browser pages seems to have been an efficient mechanism for dealing with pop-up windows. The added "browser restart" mechanism successfully re-launched a browser instance if the browser instance that was being used got closed or crashed. Those changes have all made the crawler more resilient, to the point where it is now most of the time able to complete its crawling task without any human intervention.

Another improvement in the data collection, from the first crawl to the second, was using the Ghostery browser extension rather than the Ghostery browser. This allowed to considerably broaden the number of websites successfully visited by Ghostery. It changed Ghostery from a success rate similar to Firefox's to a success rate similar to Google Chrome's. This also meant that the webdriver limitation affected only one browser, rather than two.

Despite there being limitations in the data collection, the crawl results offer multiple clear insights into behaviours that were key to the project's objectives. Indeed, results clearly indicate that Brave and Ghostery are able to significantly reduce the number of cookie banners being shown to the user. In addition, they also considerably reducing the amount of browser storage being installed and the number of requests being made. In addition to the interesting results produced by a rigorous data analysis process, this report clearly stated what could and could not be implied from the results throughout the analysis.

6 Conclusion and Future Works

The introduction mentioned certain goals of this analysis. The first one was to assess how well different the browsers being studied hid cookie banners from users. Results of this measurement indicate that Brave is able to reduce the number of visible cookie banners by 86.9%, that Ghostery reduces it by 57.7%, and that Firefox reduces it by 16.2%. Another was to assess how these browsers blocked the cookie banners. As shown in Section 4.2.3, Brave distinguishes itself by blocking, rather than hiding, cookie banners 30% more often than Ghostery and Firefox do. Indeed, Brave uses the blocking technique 71.0%, compared to 41.4% for Ghostery and 44.7% for Firefox (with the remaining value being the percentage of the time the hiding technique is used).

If a user wanted to choose a browser with a cookie banner blocking capability, and looked at its performance blocking cookie banners, reducing total and third-party storage as well as reducing total and third-party requests, then the user would probably choose Brave Browser. Indeed, in addition to performing best at hiding cookie banners, Brave also performs best reducing the number of storage sets and the number of requests being made. It reduces 78.5% of total storage compared to Google Chrome, and that reduction is of 84.2% when considering only third-party storage. Similarly, Brave has 15.0% fewer requests than Google Chrome overall, and 19.6% fewer websites visited by Brave, than by Google Chrome, register any third-party requests.

Future work could easily integrate more browsers, or extension, into the data collection. This would allow to compare the effectiveness of a larger number of tools, and assess the techniques used by them. Future work could also try to isolate the cookie banner-blocking technique as a variable, and then observe how a change in the technique used, *ceteris paribus,* leads to variations in browser storage and requests.

Moreover, it would also be relevant to perform a similar experiment using mobile devices. Indeed, mobile internet traffic represents 58.3% (53) of all total traffic, and as Merzdovnik et al. (4) showed, mobile tracking can exhibit different behaviour than desktop tracking. Therefore, it would be very relevant to analyse the performance of the tools in this new environment. This was not part of the current project due to time constraints.

More analysis could also be performed on the data collected during the crawl. For example, it could be interesting to analyse the distribution of websites when looking at storage or request data. Maybe such data representation could help bring additional insights to the data (e.g.: How skewed are the distributions?). As mentioned before, isolating the cookie banner blocking technique could also help shed light on what the impact of the techniques really is on storage and request values.

Finally, this project did not require any ethical approval. No human subjects were involved in the experiment, and no personal information was being gathered. Moreover, from a website's perspective, the crawler does not generate more traffic than any regular user. Therefore the crawler should not create any negative impact on the functioning of the websites being visited.

7 References

References

(1) Mozilla. *Client-side storage.* https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage [Accessed Sep 1, 2023].

(2) Bielova N, Legout A, Sarafijanovic-Djukic N. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. *Proceedings on Privacy Enhancing Technologies.* 2020; 2020 (2): 499-518. https://doi.org/10.2478/popets-2020-0038.

(3) Roesner F, Kohno T, Wetherall D. Detecting and defending against third-party tracking on the web. *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12),* ; 2012. pp. 155-168.

(4) Merzdovnik G, Huber M, Buhov D, Nikiforakis N, Neuner S, Schmiedecker M, et al. Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools. *2017 IEEE European Symposium on Security and Privacy (EuroS&P), 26-28 April 2017.*: IEEE; 2017. pp. 319-333.

(5) Papadopoulos P, Kourtellis N, Markatos E. Cookie synchronization: Everything you always wanted to know but were afraid to ask. *The World Wide Web Conference,* ; 2019. pp. 1432-1442.

(6) Sanchez-Rola I, Dell'Amico M, Kotzias P, Balzarotti D, Bilge L, Vervier P, et al. Can I Opt Out Yet? GDPR and the Global Illusion of Cookie Control. *Proceedings of the 2019 ACM Asia conference on computer and communications security, July 9–12, 2019.*: ACM; 2019. pp. 340-351. https://doi.org/10.1145/3321705.3329806.

(7) Libert T. An automated approach to auditing disclosure of third-party data collection in website privacy policies. *Proceedings of the 2018 World Wide Web Conference,* ; 2018. pp. 207-216.

(8) Sanchez-Rola I, Santos I. Knockin'on trackers' door: Large-scale automatic analysis of web tracking. *Detection of Intrusions and Malware, and Vulnerability Assessment: 15th International Conference, DIMVA 2018, Saclay, France, June 28–29, 2018, Proceedings 15,* : Springer; 2018. pp. 281-302.

(9) Schelter S, Kunegis J. On the ubiquity of web tracking: Insights from a billion-page web crawl. *The Journal of Web Science.* 2018; 4 .

(10) Englehardt S, Reisman D, Eubank C, Zimmerman P, Mayer J, Narayanan A, et al. Cookies that give you away: The surveillance implications of web tracking. *Proceedings of the 24th International Conference on World Wide Web,* ; 2015. pp. 289-299.

(11) The European Parliament and the Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).* 2016.

(12) Degeling M, Utz C, Lentzsch C, Hosseini H, Schaub F, Holz T. We Value Your Privacy... Now Take Some Cookies: Measuring the GDPR's Impact on Web Privacy. *Network and Distributed Systems Security (NDSS) Symposium 2019, 24-27 February 2019.*; 2019.

(13) Kulyk O, Hilt A, Gerber N, Volkamer M. "This Website Uses Cookies": Users' Perceptions and Reactions to the Cookie Disclaimer. *European Workshop on Usable Security (EuroUSEC),* ; 2018.

(14) Cofone IN. The way the cookie crumbles: online tracking meets behavioural economics. *International Journal of Law and Information Technology.* 2017; 25 (1): 38-62. https://doi.org/10.1093/ijlit/eaw013.

(15) Nouwens M, Liccardi I, Veale M, Karger D, Kagal L. Dark Patterns After the GDPR: Scraping Consent Pop-ups and Demonstrating Their Influence. *Proceedings of the 2020 CHI conference on human factors in computing systems, April 25–30, 2020.*: ACM; 2020. pp. 1-13. http://dx.doi.org/10.1145/3313831.3376321.

(16) Mozilla Web Docs. *Set-Cookie.* [Accessed June 3, 2023].

(17) Trevisan M, Stefano T, Bassi E, Marco M. 4 Years of EU Cookie Law: Results and Lessons Learned. *Proceedings on Privacy Enhancing Technologies.* 2019; 2019 (2): 126-145. https://doi.org/10.2478/popets-2019-0023.

(18) Mozilla. *Same-origin policy.* https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy [Accessed Sep 5, 2023].

(19) Mozilla. *State Partitioning.* https://developer.mozilla.org/en-US/docs/Web/Privacy/State_Partitioning [Accessed Sep 5, 2023].

(20) Randall A, Snyder P, Ukani A, Snoeren AC, Voelker GM, Savage S, et al. Measuring UID smuggling in the wild. *ACM Internet Measurement Conference (IMC '22), October 25–27, 2022.* New York, NY, USA,: ACM; 2022. pp. 230-243.

(21) Utz C, Degeling M, Fahl S, Schaub F, Holz T. (Un)Informed Consent: Studying GDPR Consent Notices in the Field. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CSS '19), November 11–15, 2019.*: ACM; 2019. pp. 973-990.

(22) Matte C, Bielova N, Santos C. Do Cookie Banners Respect my Choice? : Measuring Legal Compliance of Banners from IAB Europe's Transparency and Consent Framework. *2020 IEEE Symposium on Security and Privacy (SP), 2020.*: IEEE; 2020. pp. 791-809.

(23) Van Eijk R, Asghari H, Winter P, Narayanan A. The Impact of User Location on Cookie Notices (Inside and Outside of the European Union). *IEEE Security & Privacy Workshop on Technology and Consumer Protection (ConPro '19), 2019.*: IEEE; 2019.

(24) Yan J, Liu N, Wang G, Zhang W, Jiang Y, Chen Z. How Much Can Behavioral Targeting Help Online Advertising? *Proceedings of the 18th international conference on World Wide Web, April 20–24, 2009.*: ACM; 2009. pp. 261-270.

(25) *AdBlock.* https://getadblock.com/en/ [Accessed June 1, 2023].

(26) *EasyList.* https://easylist.to/ [Accessed June 1, 2023].

(27) *Ghostery.* https://www.ghostery.com/ [Accessed June 1, 2023].

(28) *Disconnect.* https://disconnect.me/ [Accessed June 1, 2023].

(29) *Privacy Badger.* https://privacybadger.org/ [Accessed June 1, 2023].

(30) Jueckstock J, Snyder P, Sarker S, Kapravelos A, Livshits B. Measuring the privacy vs. compatibility trade-off in preventing third-party stateful tracking. *Proceedings of the ACM Web Conference 2022,* ; 2022. pp. 710-720.

(31) Brave. *Brave Shields.* https://brave.com/shields/ [Accessed Aug 22, 2023].

(32) Ghostery. *Frequently Asked Questions.* https://www.ghostery.com/faq#general [Accessed Aug 22, 2023].

(33) Mozilla. *Firefox rolls out Total Cookie Protection by default to more users worldwide.* https://blog.mozilla.org/en/products/firefox/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/ [Accessed Aug 22, 2023].

(34) *I don't care about cookies.* https://www.i-dont-care-about-cookies.eu/ [Accessed May 23, 2023].

(35) *Consent-O-Matic.* https://consentomatic.au.dk/ [Accessed May 23, 2023].

(36) Nouwens M, Bagge R, Kristensen JB, Klokmose CN. Consent-O-Matic: Automatically Answering Consent Pop-ups Using Adversarial Interoperability. *CHI Conference on Human Factors in Computing Systems Extended Abstracts, April 29-May 5, 2022.* New York, NY, USA: ACM; 2022. pp. 1-7.

(37) Ghostery. *Introducing 'Never-Consent' by Ghostery - a new feature that removes annoying cookie popups.* https://www.ghostery.com/blog/never-consent-by-ghostery [Accessed Aug 22, 2023].

(38) DuckDuckGo. *Web Tracking Protections.* https://help.duckduckgo.com/duckduckgo-help-pages/privacy/web-tracking-protections/#cookie-consent-pop-up-management [Accessed May 23, 2023].

(39) *uBlock Origin.* https://ublockorigin.com/ [Accessed June 1 2023].

(40) Brave Privacy Team. *Blocking annoying and privacy-harming cookie consent banners* https://brave.com/privacy-updates/21-blocking-cookie-notices/ [Accessed May 23, 2023].

(41) *EasyList.* https://easylist.to/ [Accessed Aug 22, 2023].

(42) Baumgardner Niklas. *Dropping the Banner Hammer and More – These Weeks in Firefox: Issue 134* https://blog.nightly.mozilla.org/2023/04/14/dropping-the-banner-hammer-and-more-these-weeks-in-firefox-issue-134/ [Accessed Aug 1, 2023].

(43) Jueckstock J, Sarker S, Snyder P, Beggs A, Papadopoulos P, Varvello M, et al. Towards Realistic and Reproducible Web Crawl Measurements. *WWW '21, April 19–23, 2021.* New York, NY, USA: ACM; 2021.

(44) Englehardt S, Narayanan A. Online tracking: A 1-million-site measurement and analysis. *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security,* ; 2016. pp. 1388-1401.

(45) Rasaii A, Singh S, Gosain D, Gasser O. Exploring the Cookieverse: A Multi-Perspective Analysis of Web Cookies. *Proceedings of the Passive and Active Measurement Conference (PAM '23), March 21–23, 2023.*

(46) Kampanos G, Shahandashti SF. Accept All: The Landscape of Cookie Banners in Greece and the UK. *ICT Systems Security and Privacy Protection: 36th IFIP TC 11 International Conference, SEC 2021, June 22–24, 2021.*: Springer; 2021. pp. 213-227. https://doi.org/10.1007/978-3-030-78120-0_14.

(47) Célestin Matte. *Cookie glasses.* https://github.com/Perdu/Cookie-glasses [Accessed June 1, 2023].

(48) *Puppeteer.* https://pptr.dev/ [Accessed Aug 1, 2023].

(49) *Antibot.* https://bot.sannysoft.com/ [Accessed Aug 1, 2023].

(50) *Puppeteer-Extra-Plugin-Stealth.* https://github.com/berstend/puppeteer-extra/tree/master/packages/puppeteer-extra-plugin-stealth [Accessed Aug 1, 2023].

(51) Mozilla. *Navigator: webdriver property.* https://developer.mozilla.org/en-US/docs/Web/API/Navigator/webdriver [Accessed Aug 22, 2023].

(52) Mozilla. *Web-Ext.* https://github.com/mozilla/web-ext [Accessed Aug 22, 2023].

(53) Statista. *Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2022* https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ [Accessed Sep 5, 2023].

Appendix 1: Corpuses

Name	Search Terms
Short Corpus	'cookie', 'cookies', 'accept', 'reject', 'policy'
Medium Corpus	'cookie', 'cookies', 'agree', 'i agree', 'accept', 'accept all', 'accept cookies', 'i accept', 'reject', 'reject all', 'decline', 'cookie preferences', 'manage cookies', 'preferences', 'learn more', 'more information', 'privacy policy', 'privacy statement', 'cookie policy',' cookie notice', 'our partners', 'partners', 'third-party'
Long Corpus	'cookie', 'cookies','agree', 'i agree','accept', 'accept all', 'accept cookies', 'i accept', 'allow all', 'enable all', 'got it', 'allow cookies', 'reject', 'reject all', 'decline', 'mandatory only', 'required only', 'not accept', 'disable all', 'disagree', 'decline cookies', 'decline all', 'mandatory', 'optional cookies', 'essential cookies', 'non-essential cookies','strictly necessary', 'necessary cookies', 'required', 'essential', 'non-essential', 'cookie preferences', 'manage cookies', 'preferences', 'cookies options', 'consent manager', 'customize cookies', 'cookie options', 'cookies settings', 'manage settings', 'manage preferences', 'more options', 'learn more', 'more information', 'show purposes', 'further information', 'more options', 'privacy policy', 'privacy statement', 'cookie policy','cookie notice', 'our partners', 'partners', 'third party', 'vendors', 'similar technologies', 'other technologies'
Exploring The Cookieverse Corpus	'cookies', 'privacy', 'policy', 'consent', 'accept', 'agree', 'personalized', 'legitimate interest'

Appendix 2: Algorithm Schematic