Accept All Exploits: Exploring the Security Impact of Cookie Banners

David Klein*
Technische Universität Braunschweig
david.klein@tu-braunschweig.de

Marius Musch*
Technische Universität Braunschweig
m.musch@tu-braunschweig.de

Thomas Barber
SAP Security Research
thomas.barber@sap.com

Moritz Kopmann
Technische Universität Braunschweig
moritz.kopmann@tu-braunschweig.de

Martin Johns
Technische Universität Braunschweig
m.johns@tu-braunschweig.de

ABSTRACT

The General Data Protection Regulation (GDPR) and related regulations have had a profound impact on most aspects related to privacy on the Internet. By requiring the user's consent for e.g., tracking, an affirmative action has to take place before such data collection is lawful, leading to spread of so-called *cookie banners* across the Web. While the privacy impact and how well companies adhere to those regulations have been studied in detail, an open question is what effect these banners have on the *security* of netizens.

In this work, we systematically investigate the security impact of consenting to a cookie banner. For this, we design an approach to automatically give maximum consent to these banners, enabling us to conduct a large-scale crawl. Thereby, we find that a user who consents to tracking executes 45% more third-party scripts and is exposed to 63% more security sensitive data flows on average. This significantly increased attack surface is not a mere theoretical danger, as our examination of Client-Side Cross-Site Scripting (XSS) vulnerabilities shows: By consenting, the number of websites vulnerable to our verified XSS exploits increases by 55%. In other words, more than one third of all affected websites are only vulnerable to XSS *due to code that requires user consent*. This means that users who consent to cookies are browsing a much more insecure and dangerous version of the Web.

Beyond this immediate impact, our results also raise the question about the actual state of client-side web security as a whole. As few studies state the vantage point of their measurements, and even fewer take cookie notices into account, they most likely underreport the prevalence of vulnerabilities on the Web at large.

CCS CONCEPTS

• **Security and privacy → Web application security**; **Systems security**; **Browser security**.

*Both authors contributed equally to this research.

ACSAC '22, December 5–9, 2022, Austin, TX, USA

ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3564647

KEYWORDS

Web Security, GDPR, Cross-Site Scripting, Tainting

ACM Reference Format:
David Klein, Marius Musch, Thomas Barber, Moritz Kopmann, and Martin Johns. 2022. Accept All Exploits: Exploring the Security Impact of Cookie Banners. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3564625.3564647

1 INTRODUCTION

The last two decades have witnessed increasing tensions regarding the topic of online privacy. On the one hand, companies have been keen to exploit the benefits in maximizing the collection of personal data for profiling, marketing and advertising. This attitude is best represented by Scott McNeally, then CEO of Sun Microsystems who said: "You have zero privacy anyway — Get over it" [60]. On the other hand, comprehensive legal regulations have been put in place to protect the privacy of netizens. The European Union is one of the main drivers of this legal shift with the ePrivacy directive and the General Data Protection Regulation (GDPR), which came into effect in 2002 and 2018 respectively. These regulations place strict limits on when and to what degree personal data can be collected, and how long it may be stored for. To comply with data protection regulations, companies must first gain explicit consent from users before collecting their data.

A common way to obtain this consent on the Web are via so-called *cookie banners*. Thus, on a significant number of European websites, the user is greeted by a dialog querying them about what personal data the website is allowed to collect, what that data can be used for, and with whom it shall be shared. In 2019, a study by Degeling et al. [8] found that 62% of the investigated websites already had such a cookie notice. While cookie banners are supposed to give the user control over their personal data, they often employ dark patterns to lure users into giving as much consent as possible [5, 28, 34, 44] or overwhelm users with choices. For example, the cookie banner on the website of the popular German newspaper "Focus Online" (https://focus.de) queries the user about their consent to share data with 168 third party vendors, making a somewhat meaningful choice exhaustively time consuming. Thus, mainstream media mainly portraits them as annoying [4, 27, 41] and a thriving community around browser extensions blocking or removing these banners developed.

While the effects of cookie banners on web user's privacy have been well studied, an open question remains: *What is their impact on the security of users?* We propose the hypothesis that if data collection requires consent by the user, the corresponding code carrying out that collection should only be executed once consent is given. Thus, providing consent via a cookie banner dialog will increase the attack surface of a website's JavaScript code and lead to a reduction in security. This work aims to test this hypothesis by studying the effect that consent has on the loaded and executed JavaScript code and consequently the security of the including website. For this, we first develop an approach to automatically maximize the consent given to arbitrary websites. We then go on to employ dynamic taint tracking to measure the amount of security sensitive code that is executed only once consent has been given.

We show that accepting a cookie dialog leads to a increase of 63% in security related data flows. Additionally, we show that this code is indeed the cause for actual vulnerabilities, using Client-Side Cross-Site Scripting (XSS), one of the most common and severe vulnerabilities on the web, as an example. We find that giving consent significantly increases the amount of websites vulnerable to XSS by 55%. This means more than one third of all affected websites are only vulnerable to XSS due to code they execute once consent has been given.

To summarize, our contributions are the following:

- A methodology to automatically accept cookie dialogs, thereby maximizing the consent given for tracking (Section 4).
- A study over the European web landscape, measuring the effects of accepting a banner with a focus on additionally loaded and executed code (Section 5).
- An analysis on the security impact of cookie banners based on the increase of dangerous taint flows and Client-Side XSS vulnerabilities after giving consent (Section 6).

2 BACKGROUND

Cookie banners are a fairly recent phenomenon, as they are a direct consequence of regulation passed by the European Union over the last two decades. In the following, we will first give a high level overview over these regulations and the consequences for website operators. Afterwards we will provide some background on Client-Side Cross-Site Scripting (XSS), one of the most prevalent and severe vulnerabilities on the Web.

2.1 Legal Background

In 2002, the EU passed Directive 2002/22/EC [38] and 2002/58/EC [39], which are more commonly known as the "e-Privacy Directive". These directives defined the rights of the user in the realm of the Internet, and how providers of electronic communication services are supposed to handle their users' data. Then in 2009 the "e-Privacy Directive" was amended by Directive 009/136/EC [40], which is also known as the "Cookie Directive". This directive regulated online tracking through cookies and other techniques, by forcing the site providers to obtain permission before installing cookies beyond those necessary for the basic functionality of the website [16].

Subsequently, in 2016, seven years after the "Cookie Directive" came into effect, the EU adopted the General Data Protection Regulation (GDPR) [7], which became binding two years later at the end of May 2018. This regulation aims to protect personal data of European citizens, especially in the realm of the Internet. The GDPR states in Article 6 that the processing of personal data is generally prohibited, except for the cases where it is specifically allowed by law, or when the data subject, (in this case the websites' visitors), provides their explicit consent to process data for specific purposes. Thus websites started to employ so called "cookie banners" in order to comply with the GDPR. These dialogs ask the user to provide their consent for personal data processing and are a centerpiece of our study.

2.2 Cross-Site Scripting

Before we dive deeper into the world of cookie banners, we also need to introduce one of the most notorious vulnerabilities on the Web. For this, we first need to discuss one of the main security policies on the Web: the so-called *Same Origin Policy* (SOP). The aim of the SOP is to limit which resources JavaScript code can access to those from the same origin, where origin is defined as the tuple of protocol, host, and port. Thus an attacker cannot access the content of a website by, for example, simply loading it inside a hidden frame hosted elsewhere. To interact with the content of another website the attacker has to execute their code in the same origin as the target website.

The class of vulnerabilities where an attacker injects code into the origin of another website is known as Cross-Site Scripting (XSS). This vulnerability class has been known since the turn of the millennium [6]. When exploiting an XSS vulnerability, an attacker injects JavaScript code either directly or within HTML markup into a target website. This enables the attacker to execute code on the website, allowing for example the exfiltration of credentials, reading or manipulation of cookies or performing unwanted actions.

A subclass of XSS is the so-called Client-Side Cross-Site Scripting, also known as DOM-based XSS when it was first discovered in 2005 by Amit Klein [20]. Unlike traditional XSS, Client-Side XSS vulnerabilities reside purely inside client-side JavaScript code executed in the web browser. As modern websites implement more and more functionality in client-side JavaScript, the corresponding number of DOM-based XSS vulnerabilities has also increased [52]. These vulnerabilities are especially interesting as they do not require knowledge about the server-side code. As all JavaScript of a website is sent to the victim, an attacker can easily inspect the loaded code and try to find vulnerabilities. Depending on the source, it is possible that the exploit payload is never sent to the server at all, making it very difficult for the websites operator to detect attempted attacks.

The root cause for Client-Side XSS is when attacker-controllable data is used with functions which interpret test input as code and subsequently execute it. Examples for such functions, known as *sinks*, are `document.write`, `innerHTML` (both HTML sinks) or `eval`, a JavaScript sink. Several functions can act as *sources* of attacker-controllable data, most prominently the URL. An example for such a vulnerable data flow is provided in Figure 1. By cleverly crafting an exploit URL, the attacker can add arbitrary markup into the website which leads to arbitrary code execution, e.g., by including `"><script>alert(1)</script>` inside the hash. Potential victims can be lured into visiting the attacker's URL via e.g., Phishing

campains, and can be used to steal credentials from cookies or exfiltrate credentials via keyloggers.

```
let app_id = location.hash.substr(1);
// application code
document.write('<img src="https://ad.com/?referrer='
  + app_id + '">');
```

Figure 1: Data flow vulnerable to Client-Side XSS

3 COOKIE BANNERS UNDER THE HOOD

The visual interface of a cookie banner usually consists of two major parts: On one hand the consent selection, which often allows consenting to broader categories such *Functional Cookies* or *Advertisement Cookies*. Consenting and objecting to individual purposes and vendors must also be possible, but is usually hidden deeper in the UI. On the other hand, one or more buttons to accept or reject all cookies at once, or save the current selection of consents. One such typical cookie banner, in this case from *flickr.com*, is shown in Figure 2. As previous studies [12, 55] have shown, these banners often employ dark patterns such as requiring much more effort to reject than to accept, thereby nudging the user into giving consent regardless of their true intentions. In contrast, in this paper we are instead concerned about the effects that happen *after consenting to all cookies*. Before we study this, we first need to take a closer look at how a typical cookie banner implementation works under the hood.

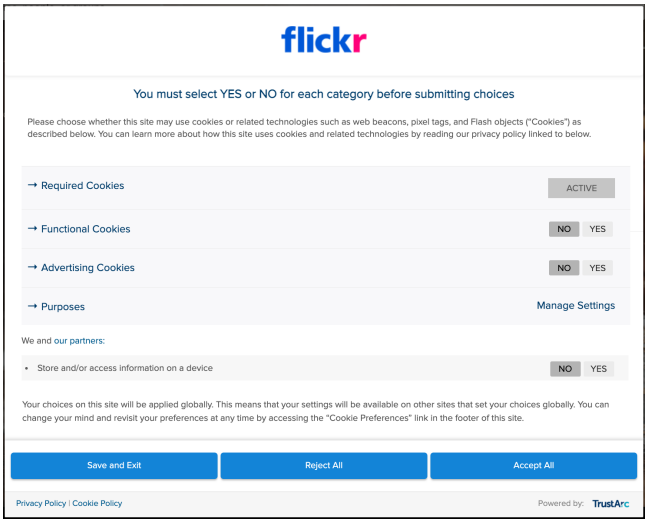

Figure 2: Example of typical cookie banner

While website owners do have the option to create their own custom implementation, it is non-trivial to create a banner that complies with all laws and regulations. Therefore, using a Consent Management Provider (CMP) has become a popular choice. These are middlemen that focus on collecting consent from the visitor and forwarding it to the advertisers. They, in turn, rely on the *Transparency & Consent Framework* (TCF) [11], which aims to help all involved parties to comply with EU laws regarding accessing and storing personal data, such as cookies. Their TCF API is what allows CMPs to propagate the consent information in a standardized manner. Thus, embedding the cookie banner provided by the CMP frees the website owner from dealing with the legal intricacies themselves.

The most important functionality that such a banner must provide is to ensure that no tracking cookies are stored and consequently no advertisements are shown to the user before they consent. In general, there are two ways to achieve this: The first is to delay the inclusion of the third-party script until consent is given as shown in Figure 3. This guarantees that the third-party is only involved once consent is given. The second is to load the third-party script from the beginning but in a state where it is running but not yet storing cookies and then only later notify the script when consent has been given, as shown in Figure 4. This latter option only works for scripts that provide an API to update the consent from within another script. Overall, this means that giving consent increases both the amount of scripts included into the page, as well as the amount of code that was already loaded but not yet executed.

```
__tcfapi("addEventListener", 2, function(tcData, success) {
  if (success && tcData.unicLoad  === true) {
      var script = document.createElement('script');
      script.setAttribute('data-ad-client', 'ca-pub-xxxxxxxx');
      script.src = 'https://pagead2.googlesyndication.com/...';
      document.head.appendChild(script);
}});
```

Figure 3: Delayed inclusion of Google Ad Manager (shortened for brevity) [56]

```
gtag('consent', 'default', {
  ad_storage: 'denied', analytics_storage: 'denied',
})
__tcfapi('addEventListener', 2, function (tcData, success) {
  if (success && tcData.unicLoad === true) {
    gtag('consent', 'update', {
      ad_storage: 'granted', analytics_storage: 'granted',
})}})
```

Figure 4: Delaying all Google tag cookies until the users consents (shortened for brevity) [57]

4 AUTOMATICALLY MAXIMIZING CONSENT

In order to analyze the difference before and after giving consent to the cookie banner in an automated fashion, we designed an approach called *Acceptify*, which automatically finds the cookie banner and its corresponding accept button on arbitrary websites. Note that in contrast to prior works on automatically interacting with cookie banners, such as *I don't care about cookies* [19], we can not just detect the banner and hide it from the user, but also need to find the correct element inside that banner that will *maximize consent* when clicked.

To achieve this, we exploit two key aspects: First, the banners have to be the point of first interaction, i.e., they can not be hidden behind other elements and need to visible at the top layer of the page. Second, they need to ask the user to consent to the tracking, i.e., they have to present a choice between multiple options in a human-readable fashion.

4.1 Acceptify: Design and Implementation

Using these insights, we can find the element that maximizes consent as follows: First, we compile a list of *candidates*, i.e., HTML elements that are likely related to cookies and giving consent. We consider all elements in the page (including all its iframes) as candidates, if they match the following three criteria:

(1) They are clickable, i.e., have the `.click` property. If the element can not be clicked, then it is not relevant for giving consent.
(2) They contain a word or phrase from our manually compiled list of 43 words across 9 different languages, with entries such as *Accept*, *Got it*, and *Permettere*.
(3) Their text content is short, i.e., consists of a maximum of 200 characters across 6 words. The rationale here is that we do not want to consider whole paragraphs of text that include these words, but only elements that conduct an action, which are usually brief and concise.

At this point, if there is only exactly one candidate, Acceptify considers this as the correct element for maximizing consent and the search stops. Similarly, if there are no candidates, Acceptify concludes that there is no cookie banner on this page and the search stops. On the other hand, if we found multiple candidates, we proceed as follows: First, we prioritize certain HTML elements over others: Buttons get the highest priority, followed by links and then div elements. If there are multiple buttons, we proceed only with those. Should there be none, we take all links etc. and if none of the candidates are of these types, we just proceed with all of them. Next, we apply multiple *filters*, i.e., try to further reduce the number of candidates until there is only one candidate. However, should a filter reduce the amount of candidates to zero, then it is ignored. The reason these checks are only filters and not strict requirements for being considered as a candidate, is that, unfortunately, not all websites adhere to the requirements of making the accept button visible at the top level within the current viewport.

Our first filter checks if only some of the candidates are visible by comparing the `offsetParent` property. After that, we check if only some of the candidates are within the current viewport, i.e., visible without scrolling by comparing `getBoundingClientRect` to the current screen dimensions. Next, we check if some of the candidates contain *bad words*, i.e., words that indicate a negative action such as *not*, *refuse*, or *nur*. Finally, we check if only some of the candidates are displayed at the top of the screen, using the code shown in Figure 5. Should, after all these filters there still be more than one candidate, we then abort the search as inconclusive. Usually however, after all these additional steps only one candidate remains, of which we are fairly confident as being the element that will maximized consent when clicked.

```
function isTopLevel(ele) {
    let {x, y} = ele.getBoundingClientRect();
    return ele === document.elementFromPoint(x,y);
}
```

Figure 5: Checking if an HTML element is visible at the top

4.2 Manual and Automated Verification

To ensure we actually click on the correct button with Acceptify, we conducted a preliminary crawl and thereby validated it twofold: On one hand, we manually verified the results using visual inspection on a random subset of the visited websites. On the other hand, we used the TCF API as an automated oracle, as we will outline in the following.

Manual visual inspection. First, we manually investigated if the selected button corresponds to what a human would identify as the "accept all cookies" button, using visual inspection. For this, we took a random sample of 250 from the 10,000 most popular websites and visited them with a modified version of our Acceptify implementation. Instead of clicking the button, the modified version visually highlights the determined candidates for the accept button using CSS rules and then creates a screenshot of the page instead of clicking the button. This way, we can quickly check for false positives and false negatives in the collected images.

Our manual inspection of the 250 images showed the following:

- True positives: On 99 websites, a banner was present and correctly detected.
- False positives: On 3 websites, no banner was present but Acceptify detected another, unrelated button.
- True negatives: On 98 websites, no banner was present and none detected.
- False negatives: On 48 websites, a banner was present but Acceptify could not detect it. On 5 of these, there was a banner present, but multiple buttons were equally likely. In the remaining cases, no button was detected, e.g., due to unsupported language or uncommon phrasing.
- Unknown: On 2 websites, the visual inspection failed due to an error and the screenshot being completely white.

While the number of false negatives is significant, this is no cause for concern in the context of our paper. Our main goal is to measure the effects of accepting a cookie banner, therefore we focus on having as few false positives as possible. This is also the reason why we do not click any button at all instead of guessing which might be the most likely, in case there are multiple viable candidates for the accept button. While our collected data on websites with a cookie banner is *not complete*, this underreporting ensures that, for the most part, the collected data is *correct*.

Using TCF as a verification oracle. Additionally, we used the TCF API [11] to obtain a machine-readable information about the consents given to the website. While the legal status of the TCF is disputed [2, 45], it is still widely deployed and helps to show the efficacy of our approach. The TCF API, if present on a website, allows us the query the consents given by the user before and after we clicked on the cookie banner and can thus be used as another

indicator if our Acceptify approach correctly interacted with the website. The current consent status can be queried from JavaScript as shown in Figure 6. If the website implements TCF correctly and Acceptify clicked the correct button, we expect the consents, which we simply represent as a sum, to increase.

```
1 __tcfapi('getTCData', 2, function(tcdata) {
2     if (!tcdata.purpose || !tcdata.purpose.consents) {
3       return -1;
4     }
5     return Object.entries(tcdata.purpose.consents).reduce((prev,
  ↪  curr) => curr[1] ? prev + 1 : prev, 0);
6 });
```

Figure 6: Querying the level of consent with the TCF API

In order to evaluate our technique, we identified 2,278 web pages from the data set used for the later experiments where Acceptify detected a cookie banner and the TCF API was present. More details on website selection can be found in Section 5. Out of those, 123 did not implement the API correctly, i.e., by not providing all the mandatory fields to query the purpose level as shown on Figure 6 on line 2 or not invoking the callback at all. For 250 clicking the button did, unfortunately, not lead to a increase in consent. Manual inspection showed these were mostly caused by either accepting something unrelated to cookies, such as notifications or a newsletter, or accidentally clicking an element that did not maximize consent, due to our lack of supporting all possible phrases in all languages spoken in the EU. However, with 1,905 of the websites correctly implementing TCF, Acceptify's interaction did lead to cookie consent, demonstrating that it performs well in a real-life setting in an overwhelming majority of cases (88.40%).

5 EFFECT OF CONSENTING COOKIES

Now that we have a reliable way to automatically accept cookies with Acceptify, we use this approach for a comprehensive study on the resulting effects of consenting to cookie banners in the wild. In order to achieve this, we conduct a large-scale study on websites within the EU as we describe in this chapter.

5.1 Experimental Setup

In the following, we will first describe our website selection and the parameters of our crawl.

Website selection. The focus of our experiment is on the European web landscape. While non-European websites might have adopted cookie banners due to the extraterritorial effects of GDPR, it is mainly a EU centric phenomenon. We thus chose to crawl a subset of the Tranco list [24]. We first excluded entries originating from *Cisco Umbrella* with the *List configurator* on the Tranco website (https://tranco-list.eu/configure) as they are DNS based and contain URLs not available to a web browser. We then took the first 1,000 entries for each EU countries TLD, generic TLDs related to the EU, as well as the UK, `.com`, `.org` and `.net`. The latter are used by both European as well as non European entities and the UK is still largely aligned with the EU from a regulatory point of view. This left us with 28,718 domains to visit. This selection is aligned with established literature, such as by Matte et al. [28]. We will make the list and the selection automatism available upon acceptance to aid reproducing our work.

Crawling setup. Our crawl took place from 20th to the 23rd of May 2022, using one server with 80 cores, 192GB RAM, and with an European IP address, on which we ran 20 worker instances in parallel. These workers based on Playwright[1] in version 1.21, which we modified with our custom crawling implementation that we will describe in more detail in the next subsection. For our browser we used "Project Foxhound" in version 96.0.3 [2], a Firefox fork enhanced with taint-tracking for both SpiderMonkey, its JavaScript engine, as well as Gecko, Firefox's rendering engine. This allows us to capture data flows related to potential security issues. Exploit generation and validation were done in parallel with the regular crawling, to minimize the time between the initial visit and the validation. As websites tend to change frequently, taint flows are fairly volatile. So in addition to the regular crawling, we had 8 validation workers running in parallel.

5.2 Implementation

On each website, we waited for the load event for a timeout of up to 30 seconds, otherwise we flag the website as unavailable and move on. Once the website did load successfully, our crawler waits for an additional 5 seconds before interacting with it. This gives the website more time to load additional dynamic content, such as the cookie banner itself. From there, our implementation proceeds as follows: First, we execute Acceptify in the main frame, as well as all iframes, if there are any. We then merge the results of all these executions and check the number of elements that Acceptify has returned. If there are none or multiple, we stop here as there is either no banner or it is not entirely clear which element would maximize consent. If Acceptify detected exactly one element that will maximize consent, it clicks this element and waits for another 10 seconds to allow the website to load dynamic content, such as ads, based on the newly given consent. Moreover, if we detected and accepted a cookie banner on the landing page with Acceptify, we now queue an additional 20 links to subpages within the same domain. This allows us to detect data flows which rely on e.g., URL parameters being present and is important to increase our coverage, e.g., as landing pages rarely make use of `location.hash`. Finally, we also serialize and store all cookies before and after interacting with the cookie banner, as this allows us the verify the security impact of giving consent, as we will describe in the next section.

5.3 Results

Out of the list of 28,718 landing pages, we successfully visited 23,113 of them. 1,780 sites failed due to network errors, such as DNS resolution errors or a refused connection. Moreover, 1,133 sites did not successfully load within our timeout, causing our visit to abort. Another 888 sites returned an HTTP error in the 4XX or 5XX range. Finally, 1,804 sites failed for various other reasons, such as crashing the crawler or taint browser. Including all sub pages, we visited 176,273 pages in total across these 23,113 sites.

[1] Available at https://playwright.dev
[2] Available at https://github.com/SAP/project-foxhound/releases/tag/v96.0.3

Out of the successfully visited sites, Acceptify detected and clicked on a consent button on 8,149 (35.26%) of them. Figure 7 shows how the selected websites are distributed unevenly across the Tranco ranking, as our selection approach creates a bias towards the lowest ranks, i.e., the more popular websites. However, as the figure also shows, these are also the most likely to employ a cookie banner, with banners on almost half of all sites in the top 100k compared to less than one fifth of the sites in the higher ranks.

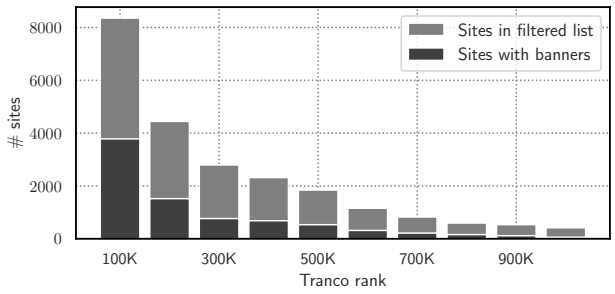

Figure 7: Distribution of websites and cookie banners across the Tranco ranking. Due to our filtered list of 23,113 successfully visited sites with a focus on EU content, these sites are not evenly distributed.

In the following, we describe the resulting effects of consenting to cookies in more detail, focusing on two aspects: The effect on the number of stored cookies and on the amount of loaded JavaScript code. For all these analyses, we only consider the 8,149 sites where we found and interacted with a consent button. The results for these two analyses are summarized in Table 1, which we will describe in more detail in the following.

Table 1: Impact of Acceptify on cookies and scripts

Resources		Sites	Initial	Accept	Increase
Cookies	First Party	8,085	71,538	119,318	66.79%
	Third Party	7,181	38,260	167,814	338.61%
Scripts	First Party	7,699	97,953	98,739	0.80%
	Third Party	7,931	117,165	169,352	44.54%

Effect on stored cookies. First of all, we studied the number of stored cookies before and after consenting to cookies, as another validation that our Acceptify approach works as expected. Initially, these 8,149 sites did set a total of 109,798 cookies during the first load of the page, i.e., each site does set an average of 12.70 cookies without any interaction by the user at all. The majority of them were first-party cookies, with 8.27 from a first party on average compared to 4.42 third-party cookies on average. After running Acceptify and thus maximizing consent, these numbers changed to a total of 287,132 cookies, i.e., an increase of over 20 additional cookies per site on average. It is also noteworthy that the number of first-party cookies increased only by 66.79% from 71,538 to 119,318 cookies, while the number of third party cookies increased by 338.61% from 38,260 to 167,814 cookies. We see this as another demonstration that Acceptify works as expected, as the number of cookies, and especially the number of third party cookies that typically require consent by the user, did increase significantly after our interaction.

Effect on loaded scripts. Looking at the number of loaded scripts within the main frame of the website as a first approximation for executed code, we see an even clearer effect towards more third party inclusions after consenting to cookies. At first, loading our set of 8,149 sites without any interaction caused the inclusion of 215,118 scripts, i.e., an average of 24.87 scripts per website. These are roughly evenly distributed across first- and third-party scripts, with 11.33 first- and 13.55 third-party scripts on average per site. After consenting to cookies the total number of scripts increased to 268,091 or an average of 31.13 scripts per site. However, the number of first-party scripts increased only very slightly from 97,953 to 98,739 scripts. On the other hand, the number of third-party scripts significantly increased from 117,165 to 169,352 scripts. In other words, while accepting cookies has almost no impact on the amount of first-party code that is executed, the simple action of consenting cookies increases the amount of third-party code by 44.54%. As we will investigate next, this increase and code and thus attack surface has a considerable impact on the client-side security of the affected web applications.

6 SECURITY IMPACT

As highlighted in Section 5.3, consenting to data collection by clicking accept on a cookie banner has a profound impact on the amount of code a website loads and executes. As these additional JavaScript files are loaded in the security domain of the main website, this can weaken the security of the including website. While the additional code is required, e.g., to enable data collection, it might also expose the user to vulnerabilities if said code was written in an insecure fashion. By using a taint enabled browser, we can also measure data flows between security sensitive sources and sinks to get an idea about the amount of security sensitive code a user is exposed to. Dynamic taint tracking allows us to capture a complete view of the effect consenting has on the execution of JavaScript code. Not only do we detect data flows in newly included scripts, we also detect data flows located in previously loaded scripts but only now triggered by the user's interaction.

6.1 Taint Flows

The mere number of included scripts is not necessarily representative of the amount of actually executed code. Therefore, we also report on the increase in security relevant taint flows as a result of accepting the cookie banner. Security sensitive taint flows are data flows where the source is attacker-controllable data and ends up in a sink related to client-side security issues. Not every website where Acceptify interacted with a cookie banner contains security sensitive data flows. However, with 7,577 out of the 8,149 sites, the vast majority of 92.98% websites do so. We then divided these security relevant taint flows into three categories, as described in the following:

(1) Reflected: The tainted value is directly written into the website, either in dynamically evaluated JavaScript code or into

the DOM. The value is thus reflected back to the user, causing direct code execution. The number of taint flows in this category increases from 455,258 to 746,414, by 60.99%. Especially worrisome is the large increase of data flows into JavaScript sinks, as writing a sanitizer for such sinks is especially difficult due to the syntactical complexity of JavaScript.

(2) Generic: Here the attacker-controllable data is used as input for a postMessage, allowing the attacker to potentially influence cross origin communication and attack resources embedded into the website. This category sees the sharpest increase, from 2,625 to 5,863, by 123.35%.

(3) Stored: The tainted value is written into the browser's storage, either as a cookie or into localstorage. This can be used to achieve stored XSS or to manipulate the users profile for example. Here the number of taint flows increased from 38,167 to 56,685, by 67.33%.

In total, the amount of security relevant taint flows increases from 496,050 to 808,962, i.e., by 63.08%. This highlights a significant increase in security sensitive code which is inserted and executed in the including page's origin upon consenting to data collection. Thus, allowing data collection automatically exposes the user to additional security risks, they might be unaware of. A comprehensive breakdown of the taint flow statistics is provided in Table 2.

Table 2: Impact of Acceptify on recorded taint flows

		Sites	Initial	Accept	Increase
Taint Flows		7,577	496,050	808,962	63.08%
Reflected XSS:	URL → HTML	452	1,970	2,657	34.87%
	URL → JavaScript	112	1,480	3,024	104.32%
	URL → URL	7,474	451,808	740,733	63.95%
Generic	URL → postMessage	499	2,625	5,863	123.35%
Stored XSS:	URL → cookie	2,645	20,444	31,942	56.24%
	URL → LocalStorage	1,542	17,723	24,743	39.61%

6.2 Exploit Generation and Verification

Each additional reported taint flow increases the exposure to security risks. If one data flow lacks proper sanitization, i.e., ensuring no harmful characters pass through to the sink, the website as a whole becomes insecure. However, not all security relevant taint flows result in exploitable vulnerabilities. They might either employ sanitization or the surrounding application logic might ensure no code execution takes place, e.g., by parsing the tainted value as a number.

Thus, to measure the security impact of the additional code that is executed after accepting a cookie banner in terms of actual vulnerabilities, we generate proof-of-concept exploits and validate them. For this, we focus on Client-Side XSS, as it consistently ranks among the most common and severe security risks for web applications over the last decade [35–37]. Our XSS exploit generation strategy is in line with prior work [25, 29, 53], thus we will only give a short overview here. For a more detailed discussion on generating exploit URLS for Client-Side XSS, we refer the reader to these works.

Generally speaking, our exploit generator produces an injection-context-specific *breakOut* sequence, a sink-specific *payload*, and an injection-context-specific *breakIn* sequence. For example, looking back at Figure 1 where the attacker controlled data ends up in a double-quoted attribute of an `img tag`. To break out of this context, we first close the attribute with the corresponding quote type and then close the tag itself. The resulting *breakOut* is thus `">`. Next, we generate a payload suitable for the sink function. To detect that code execution did occur, an alert box inside a script code is generated as the payload in this case. The *breakIn* sequence aims to return the parser to a state where it does not cause an error due to the now dangling characters that originally closed the `img` tag. For the HTML context this is not necessary, as the HTML parser tolerates the dangling `">` suffix, which is hard coded after the injection point. Thus for the given example, the exploit payload would look as follows: `"><script>alert('XSS')</script>`.

If the injection takes place in an attribute context of an HTML element where code execution without user interaction is possible, e.g., `onload` or `onerror`, the payload is simpler. Instead of breaking out of the enclosing tag, we first break out by the current attribute with the corresponding quote, add the event handler to the tag and lastly reenter an attribute. For example, for a double-quoted attribute, we would generate the following exploit: `" onload=alert('XSS') onerror=alert('XSS') x="`. This makes the generated exploits highly specific to the injection context encoded in the taint flow.

Our approach diverges from prior work when validating the exploit URL: In addition to the raw taint flows, we store the current page's cookie jar with every flow. Then, when validating a Client-Side XSS exploit URL, we visit the URL twice: once without cookies and once with the cookies stored with the taint flow. If the vulnerability relies on the cookie banner being accepted, only the second visit should result in code execution. Exploit validation is done with legacy URL encoding, in line with previous works [21, 25, 29, 50].

We only measure the vulnerabilities occurring in the first party, i.e., the website we directly visit. Vulnerabilities in third party frames are out of scope for our work, as exploiting the actual website is a lot more involved, requiring e.g., postMessage.

6.3 Discovered Vulnerabilities

Of the 8,149 websites where Acceptify successfully interacted with a cookie banner, our exploit generation strategy was able to generate an exploit on 1,395. Out of those, we found 73 domains to be directly vulnerable to Client-Side XSS. That is, their vulnerabilities are independent from consenting to cookies, as the vulnerability resides in the code providing regular functionality of the website. However, on 44 out of the investigated domains we confirmed XSS exploits that require consenting to cookies, i.e., the consent did not only increase the available attack surface but did result in additional vulnerabilities introduced into the website. Moreover, 9 of those domains contain both vulnerabilities requiring cookies and vulnerabilities that do not.

In other words, over one third of the domains with Client-Side XSS are *only vulnerable if the user has given consent to data collection.* This increase of 55% closely aligns with the increase of security sensitive data flows (63.08%), highlighting their the significant impact on security. Thus, without giving prior consent, an attackers

capability of exploiting a user is greatly reduced. This means that attacks without user interaction, e.g., the common approach of loading the vulnerable website in a hidden iframe, would fail unless the victim had given consent on a previous visit.

Looking at the distribution of XSS vulnerabilities across the website ranks in Figure 8, we find that their distribution is similar to the one shown in Figure 7, as websites ranked higher are more likely to employ tracking and thus a cookie banner. Consequently, they are more likely to be affected by the security issues studied in this work. This again highlights the high relevance of our findings, as the most popular websites are most often affected by these vulnerabilities, but at the same time also represent the most valuable targets for attackers.

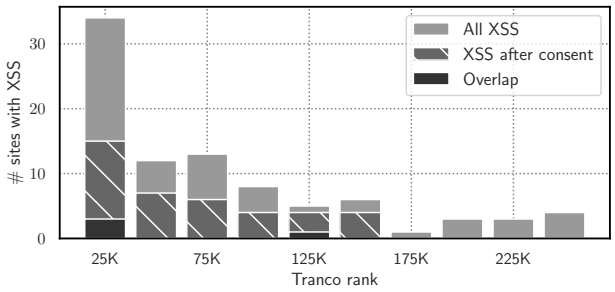

Figure 8: Vulnerabilities across the Tranco ranking. Truncated to the first 250,000 entries for readability. Outside of this range are 1 domains with XSS after consent, 17 domains with direct XSS and 3 domains containing both.

7 DISCUSSION

The presented results highlight the significant impact that privacy regulations have on the security of websites. An increase of security sensitive data flows by almost 65% significantly raises the risk for web users. As we have shown, due to these additional flows, we were able to exploit a considerable amount of websites *that would have been secure otherwise.* In the following, we make a call to action for both browser vendors and security researchers to improve this situation. Moreover, we discuss our limitations, potential future work, as well as ethical considerations.

7.1 Calls to Action

In the following we use our results to motivate fine-grained browser security policies and more awareness of the impact of cookie banners in security studies.

Call for effective countermeasures. We see our findings as a call to action for browser vendors. The root cause for the increase in security related data flows is that additional code is executed inside the security perimeter of the *including website.* Thus, every vulnerability in such code, loaded from third parties and thus out of control of the website operator, also affects the including website. Browsers already support mechanisms to minimize the attack surface by isolating such third-party code, e.g., by loading it into an sandboxed iframe. However, this strict isolation hinders many use cases such as advertisements and tracking, which rely on accessing all of the user's interaction with the website and not just interactions within the iframe. Thus, most ad providers still expect their code to be included directly into the website.

The underlying issue is that the browser currently does not support a way to provide finely grained security policies, i.e., to accurately describe what a certain piece of code can do and restrict all other actions. Since no such mechanism exists, this gives the included code full control over the website. Several countermeasures have been proposed to counteract this problem, some of which we will discuss in our related work. However, none of these have found wide-spread adoption by browsers or advertising providers. While the latter have few incentives to limit what they can do, the browser vendors should be concerned about their users' security. We thus encourage browser vendors to finally adopt a permission model that finds a good compromise between usability for third-party code providers and security for their users. The recently published postmortem of failed solutions [48] provides a great starting point to steer future permission models into the right direction.

Call for incorporation during research. Additionally, we also see this as a call to action for web security researchers to incorporate these findings into future studies. As we found, crawling from within the EU can have a considerable impact on the default security level of a website, i.e., more sites appear to be secure than they are as their vulnerabilities are hidden behind a consent banner. This means, to accurately measure the prevalence of client-side web vulnerabilities, researchers should either select a vantage point that is not affected by strict privacy regulations or adopt similar strategies as we presented with Acceptify, to take cookie notices into account. Unfortunately, according to Demir et al. [9] only 25% state the vantage point of their crawls in their papers. This means we have to assume that the actual state of the security landscape on the Web is most likely even worse than previously assumed, as all data collections over the last years from within the EU are affected and would *underreport on vulnerabilities.* In order to improve the situation by accelerating the adoption of cookie-aware security studies, we plan to make the source code of Acceptify available on publication of this paper.

7.2 Future Work

In the following, we briefly discuss some limitations of our work and promising research directions for future work in this space.

Automatically interacting with banners. To support a large-scale study on the security impact of consenting cookies, we designed an automatic approach to maximize consent called Acceptify. While we have shown that our Acceptify approach performs reasonably well, there are nevertheless cases in which it could perform better. For once, the EU has 24 official languages, making complete support for all consent related phrases in all languages challenging. Therefore, we focused only on common phrases in the most common languages. However, this is not a general limitation of our approach and could be improved with further refinements and feedback from native speakers. Moreover, there are many edge-cases, such as sites that accidentally display multiple banners or have one in their DOM, but do not display it, in which case the current implementation does not take any action. Overall, it is important to

note for the scope of this paper there was no need for Acceptify to be complete as our main goal was to highlight the differences before and after consenting to cookies. In other words, the exclusion of some banners (due to e.g., unsupported languages) will have no effect on our numbers beyond reducing our sample size. One thing to keep in mind for future research is that the number of cookie banner implementations and frameworks could become further fragmented, thus complicating automatic interactions. This is due to the decision that the TCF has been ruled unlawful [2], which was one of the largest consent management providers at the time of writing.

Precise attribution of parties. Our data analysis relies on separating cookies and scripts into originating from either a first- or a third-party. However, correctly attributing domains to parties is still an somewhat unsolved problem, especially when dealing with large companies that own many domains. For example, consider a script inclusion from `google.com` to `youtube.com`, or from `facebook.com` to `fbcdn.com`. While the domains are not related on any hierarchy, both these examples might still actually represent a first-party inclusion as both are owned and operated by the same company. One proposed solution for this is the notion of an *extended Same Party* [49], which can deal with many of these issues expanding the set of domains that are considered to be the same party. A downside of this technique is that it relies on various heuristics to group related domains. On the other hand, the draft for *First-Party Sets* [59] would allow websites to describe which other domains should be considered first-party to them in a mutual fashion. Should this standard find wide-spread adoption, it would allow an even more precise attribution of parties in the future.

Crawling coverage. Another limitation of our study is that due to time and resource constraints, we can only visit a limited amount of sub-pages on each domain. Therefore, might have missed security relevant taint flows and consequently XSS vulnerabilities. As such, our results should only be seen as a *lower bound* that demonstrates the relative difference between the amount of vulnerabilities before and after consenting to cookies. There are already multiple possible research efforts in progress to increase this crawling coverage beyond merely crawling for longer and with more resources. On one hand, Eriksson et al. [10] demonstrate how a data-driven approach can tackle these crawling challenges in a generic fashion. On the other hand, there is also research on specific sub-problems that hinder this so-called *deep crawling* of web applications. For example, Jonker et al. [17] recently published their approach to enable *post-authenticated crawling*, by automatically registering and logging in on arbitrary websites. By making use of these and similar techniques, future work could paint a more complete picture of the prevalence of these XSS vulnerabilities in the wild.

Exploit generation and validation. While the general techniques to automatically generate Client-Side XSS exploit URLs are well studied and understood [25, 29], they only cover basic cases. Taking into account more of the application logic can lead to more discovered vulnerabilities. The named approaches only support a small subset of JavaScript functions for for exploit generation, such as encoding functions. A more complete view of the operations on the tainted string, such as ensuring JSON formatted values stay valid or placing the exploit inside encoded values might improve the detection rate.

On a similar note, input sanitization is not taken into account in our study. Incorporating automated sanitizer security analysis and transformation of the payload to bypass an insecure sanitizer, such as the technique presented by Klein et al. [21] would likely improve the vulnerability detection rate.

7.3 Ethical Considerations

Probing a website for vulnerabilities involves interacting with the infrastructure of its operator. To avoid any service interruptions or harm for their users, we selected Client-Side Cross-Site Scripting as the vulnerability class to evaluate. As this vulnerability purely resides in the client-side JavaScript code, we can simulate both the attacker and the victim. In other words, injected code is executed in the browser and in the majority of cases never sent to the server. Additionally, the chosen payload to detect code execution was carefully chosen such that it is specific to our setup. Thus, even if the website somehow turns the vulnerability into e.g., a stored XSS, the payload will not interrupt the websites functionality. Moreover, we are planning to contact all affected domain owners to help them resolve their issues before making our findings public.

8 RELATED WORK

The related work can be roughly divided into three parts, studies focused on cookie banners, works related to client-side vulnerabilities, and existing countermeasures trying to solve this issue.

8.1 Cookie Banners

Since the GDPR went into effect in 2018, there have been many academic works on both its legal as well as privacy aspects. For example, Hils et al. [12] studied the ecosystem of Consent Management Providers (CMPs) and found their adoption doubled twice each year between 2018 to 2020. Sanchez-Rola et al. [43], on the other hand, recently published a more general investigation into the whole cookie ecosystem and all its actors, not limited to CMPs. Sanchez-Rola et al. [42] focused on the tracking on high-traffic websites from inside and outside the EU and compared the information presented to the user and the actual tracking implemented through cookies. While they found that sites started to reduce tracking due to third-party opt-out services, tracking was still ubiquitous as they found cookies that can identify users on more than 90% of the sites in their dataset.

Following on that, Santos et al. [44] defined 22 requirements for cookie banners, while Matte et al. [28] tried to identify potential legal violations of TCF cookie banners, which are banners that follow IAB Europe's Transparency and Consent Framework [11]. During their tests they found suspected violations in 54% of websites checked. Nouwens et al. [34] studied the impact of a banners design on the choices users make regarding consent. Finally, Bollinger et al. [5] used machine learning to automatically interact with cookie banners in order to discover GDPR violations at scale, finding at least one violation in almost 95% of all websites they analyzed.

Browser extensions such as Consent-O-Matic [22] and Cliqz Autoconsent [26] already exist to automatically provide a user's cookie policies. In contrast to Acceptify, however, these techniques rely on

detection of DOM elements of supported CMPs. CookieBlock [5] is a similar extension which filters cookies which violate a user's global privacy settings, but does not automate cookie banner interaction. As mentioned earlier, the *I don't care about cookies* [19] extension simply hides cookie banners from users, but does not attempt to apply an overall privacy policy.

Overall, in contrast to these previous publications that focused on the privacy and legal aspects of cookie banners and the GDPR, our paper investigates an orthogonal issue, i.e., the security implications of consenting to these banners in the context of XSS.

8.2 Client-Side Vulnerabilities

A wide range of academic work has studied the prevalence of client-side vulnerabilities on the Web. First of all, the usage of outdated JavaScript libraries [23] and the security consequences of including third-party code [13, 33, 49] have seen significant academic attention over the last years. Moreover, there also many other relevant vulnerabilities such as attacks abusing insecure PostMessage handlers [47, 51] or Prototype Pollution [18].

Yet, of all these web vulnerabilities, Cross-Site Scripting has received the most academic attention due to being a vulnerability that is both ubiquitous and severe and is consequently also the focus of our work. Looking back to 2013, Lekies et al. [25] first used a taint-tracking enabled browser to detect security relevant data flows on websites, automatically crafted exploit payload URLs and validated them to study the prevalence of Client-Side XSS. Later works have built on their methodology, improving various aspects such as the exploit generation [3, 29, 53] as well as adding support for persistent variants [50].

In addition to studying the prevalence of Client-Side XSS, additional aspects have been researched. For example, Stock et al. [52] studied the evolution of Client-Side XSS, highlighting how Client-Side XSS rose to prominence together with the Web 2.0. To gain a deeper understanding on how such vulnerabilities occur, Stock et al. [53] have first investigated the complexity of the vulnerable code resulting in code execution. Later Klein et al. [21] investigated the approaches websites take to protect themselves against XSS.

Our work builds on these listed insights. However, to the best of our knowledge, the security benefits of cookie banners and the resulting risks of giving consent to these banners with respect to XSS have not yet been studied.

8.3 Existing Countermeasures

Over the years there have been many attempts to allow the inclusion of third-party code without compromising the security of the including site itself, which would prevent the vulnerabilities that we discuss in this work. First of all, multiple approaches that modify the browser to enforce security policies were proposed, such as BEEP [15], Conscript [30] and WebJail [58]. However, lacking support by browser vendors, these have not found widespread use. On the other hand, there were also approaches to create a safe subset of JavaScript, like Caja [31], JSand [1], and Treehouse [14]. While their implementation does not require changes to the underlying browser, it does mean a rewrite of all untrusted code. Lacking support from third-parties such as advertisement networks, these approaches likewise struggled to solve the underlying problem.

Beyond that, there were also academic publications on defensive mechanisms that need neither browser modifications nor support by third-parties. For example, Ter Louw et al. [54] proposed AdJail, an isolation framework specifically designed for advertisements, while Musch et al. [32] proposed ScriptProtect, a mechanism that prevents the introduction of Client-Side XSS through benign-but-buggy third parties such as advertisement providers. Finally, Snyder et al. [46] built a browsing extension that allows to selectively disable DOM features, in an attempt to reduce the attack surface by disabling features which are not needed by a website. However, despite all these efforts, insecure third party code is still a significant contributor to vulnerabilities even today, as our results have shown.

9 CONCLUSION

In this paper we have tested the hypothesis that accepting cookie banners will lead to an increased security risk for users. The argument being that consenting to tracking should lead to an increase in the amount of additional code loaded in, and executed on the including website. As each additional fragment of code can introduce vulnerabilities affecting the website as a whole, this can have a significant impact on the security of all visitors.

To explore this phenomenon, we designed an approach that automatically maximizes consent on all visited websites and used this to perform a large-scale study over the European web landscape. We then used this setup to evaluate the effect that accepting a cookie dialog has on the security of the loaded and executed code. We found that giving consent to these banners indeed leads to 45% more third-party scripts being included into the site. Moreover, the number of security sensitive data flows, i.e., flows that are potentially exploitable, increases by an even larger margin of 63%. To highlight the actual security impact of this additional code with concrete vulnerabilities, we generated proof-of-concept exploits using Client-Side XSS as an example. Thereby, we found that the number of websites vulnerable to XSS increases by 55% after consenting. This means that more than one third of all affected websites are only vulnerable to XSS due to the code they execute after consent has been given.

In conclusion, the initial question – does consenting to cookie banners have a negative security impact – can be answered with a resounding yes. We found that consenting significantly increases both the theoretical attack surface, as well as the actual exposure to concrete XSS vulnerabilities. Our results motivate two calls to action: firstly for browser vendors to implement more effective countermeasures against vulnerabilities in third-party code, and secondly to security researchers to ensure they do not ignore the security impact of cookie banners and effectively underreport the prevalence of vulnerabilities on the web.

ACKNOWLEDGMENTS

We would like to thank all anonymous reviewers for their valuable comments and suggestions. Moreover, we gratefully acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. This work has also received funding from the European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206.

REFERENCES

[1] Pieter Agten, Steven Van Acker, Yoran Brondsema, Phu H Phung, Lieven Desmet, and Frank Piessens. 2012. JSand: complete client-side sandboxing of third-party JavaScript without browser modifications. In *ACSAC*.

[2] Belgian Data Protection Authority. 2022. Concerning: Complaint relating to Transparency & Consent Framework. https://www.autoriteprotectiondonnees.be/publications/decision-quant-au-fond-n-21-2022-english.pdf. Accessed 27.06.2022.

[3] Souphiane Bensalim, David Klein, Thomas Barber, and Martin Johns. 2021. Talking About My Generation: Targeted DOM-based XSS Exploit Generation using Dynamic Data Flow Analysis. In *Proceedings of the 14th European Workshop on Systems Security*. ACM.

[4] Bianca Ferrari. 2021. 'It's Bad Design On Purpose' – Why Website Cookie Banners Look Like That. Online https://www.vice.com/en/article/m7epda/its-bad-design-on-purpose-why-website-cookie-banners-look-like-that. Accessed 27.06.2022.

[5] Dino Bollinger, Karel Kubicek, Carlos Cotrini, and David Basin. 2022. Automating cookie consent and GDPR violation detection. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association.

[6] Cert/CC. 2000. CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=496186. Accessed 09.04.2021.

[7] Council and Parliament of European Union. 2016. 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). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504.

[8] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. 2019. We Value Your Privacy ... Now Take Some Cookies: Measuring the GDPR's Impact on Web Privacy.. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*. https://doi.org/ndss-paper/we-value-your-privacy-now-take-some-cookies-measuring-the-gdprs-impact-on-web-privacy/

[9] Nurullah Demir, Matteo Große-Kampmann, Tobias Urban, Christian Wressnegger, Thorsten Holz, and Norbert Pohlmann. 2022. Reproducibility and Replicability of Web Measurement Studies.. In *Proc. of the International World Wide Web Conference (WWW)*. 533–544. https://doi.org/10.1145/3485447.3512214

[10] Benjamin Eriksson, Giancarlo Pellegrino, and Andrei Sabelfeld. 2021. Black widow: Blackbox data-driven web scanning. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1125–1142.

[11] IAB Europe. 2021. What Is The Transparency & Consent Framework (TCF)? https://iabeurope.eu/transparency-consent-framework/.

[12] Maximilian Hils, Daniel W Woods, and Rainer Böhme. 2020. Measuring the emergence of consent management on the web. In *Proceedings of the ACM Internet Measurement Conference*. 317–332.

[13] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. 2019. The chain of implicit trust: An analysis of the web third-party resources loading. In *The World Wide Web Conference*.

[14] Lon Ingram and Michael Walfish. 2012. Treehouse: Javascript Sandboxes to Help Web Developers Help Themselves.. In *USENIX ATC*.

[15] Trevor Jim, Nikhil Swamy, and Michael Hicks. 2007. Defeating script injection attacks with browser-enforced embedded policies. In *WWW*.

[16] Bobbie Johnson. 2012. What you need to know about the EU Cookie Law. https://gigaom.com/2012/05/25/cookie-law-explainer/.

[17] Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and Marc Sleegers. 2020. Shepherd: a generic approach toautomating website login?

[18] Zifeng Kang, Song Li, and Yinzhi Cao. 2022. Probe the Proto: Measuring Client-Side Prototype Pollution Vulnerabilities of One Million Real-world Websites. (2022).

[19] Daniel Kladnik. 2020. I DON'T CARE ABOUT COOKIES 3.2.4. https://www.i-dont-care-about-cookies.eu/.

[20] Amit Klein. 2005. DOM Based Cross Site Scripting or XSS of the Third Kind. *Web Application Security Consortium, Articles* (2005).

[21] David Klein, Thomas Barber, Souphiane Bensalim, Ben Stock, and Martin Johns. 2022. Hand Sanitizers in the Wild: A Large-scale Study of Custom JavaScript Sanitizer Functions. In *2022 IEEE European Symposium on Security and Privacy (EuroS&P)*. 236–250.

[22] Rolf Bagge Janus Bager Kristensen. 2019. Consent-O-Matic. Online https://github.com/cavi-au/Consent-O-Matic. Accessed 27.06.2022.

[23] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web.. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*. https://doi.org/ndss2017/ndss-2017-programme/thou-shalt-not-depend-me-analysing-use-outdated-javascript-libraries-web/

[24] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation.. In *NDSS*.

[25] Sebastian Lekies, Ben Stock, and Martin Johns. 2013. 25 Million Flows Later: Large-scale Detection of DOM-based XSS.. In *ACM CCS*.

[26] Sam Macbeth. 2020. Cliqz Autoconsent. Online https://github.com/ghostery/autoconsent. Accessed 27.06.2022.

[27] Matt Burgess. 2021. How to bypass and block infuriating cookie popups. Online https://www.wired.co.uk/article/cookie-popup-blocker-gdpr. Accessed 27.06.2022.

[28] Célestin Matte, Nataliia Bielova, and Cristiana Santos. 2020. Do Cookie Banners Respect my Choice?: Measuring Legal Compliance of Banners from IAB Europe's Transparency and Consent Framework. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 791–809.

[29] William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, and Limin Jia. 2018. Riding out DOMsday: Towards Detecting and Preventing DOM Cross-Site Scripting.. In *NDSS*.

[30] Leo A Meyerovich and Benjamin Livshits. 2010. ConScript: Specifying and enforcing fine-grained security policies for Javascript in the browser. In *Oakland*.

[31] Mark S Miller, Mike Samuel, Ben Laurie, Ihab Awad, and Mike Stay. 2008. Safe active content in sanitized JavaScript. *Google, Inc., Tech. Rep* (2008).

[32] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. 2019. Scriptprotect: mitigating unsafe third-party javascript practices. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*.

[33] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*.

[34] Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. 2020. *Dark Patterns after the GDPR: Scraping Consent Pop-Ups and Demonstrating Their Influence*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376321

[35] OWASP Foundation Inc. 2013. OWASP Top 10 – 2013 – The Ten Most Critical Web Application Security Risks. https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf. Accessed: 16.09.2021.

[36] OWASP Foundation Inc. 2017. OWASP Top 10 – 2017 – The Ten Most Critical Web Application Security Risks. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Accessed: 23.07.2021.

[37] OWASP Foundation Inc. 2021. OWASP Top 10 – 2021. https://owasp.org/Top10/. Accessed: 16.09.2021.

[38] European Parliament and the Council. 2002. Directive 2002/22/EC of the European Parliament and of the Council. https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:32002L0022.

[39] European Parliament and the Council. 2002. Directive 2002/58/EC of the European Parliament and of the Council. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32002L0058.

[40] European Parliament and the Council. 2002. Directive 2009/136/EC of the European Parliament and of the Council. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32009L0136.

[41] Privacy International. 2019. Most cookie banners are annoying and deceptive. This is not consent. Online https://privacyinternational.org/explainer/2975/most-cookie-banners-are-annoying-and-deceptive-not-consent. Accessed 27.06.2022.

[42] Iskander Sanchez-Rola, Matteo Dell'Amico, Platon Kotzias, Davide Balzarotti, Leyla Bilge, Pierre-Antoine Vervier, and Igor Santos. 2019. Can I Opt Out Yet? GDPR and the Global Illusion of Cookie Control. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 340–351.

[43] Iskander Sanchez-Rola, Matteo Dell'Amico, Davide Balzarotti, Pierre-Antoine Vervier, and Leyla Bilge. 2021. Journey to the center of the cookie ecosystem: Unraveling actors' roles and relationships. In *IEEE Symposium on Security and Privacy*.

[44] Cristiana Santos, Nataliia Bielova, and Célestin Matte. 2020. Are cookie banners indeed compliant with the law? : Deciphering EU legal requirements on consent and technical means to verify compliance of cookie banners. *Technology and Regulation* 2020 (Dec. 2020), 91–135. https://doi.org/10.26116/techreg.2020.009

[45] Cristiana Santos, Midas Nouwens, Michael Toth, Nataliia Bielova, and Vincent Roca. 2021. Consent Management Platforms Under the GDPR: Processors and/or Controllers?. In *Privacy Technologies and Policy*. Springer International Publishing, 47–69.

[46] Peter Snyder, Cynthia Taylor, and Chris Kanich. 2017. Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 179–194.

[47] Sooel Son and Vitaly Shmatikov. 2013. The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites.. In *NDSS*. https://doi.org/ndss2013/postman-always-rings-twice-attacking-and-defending-postmessage-html5-websites

[48] Steven Sprecher, Christoph Kerschbaumer, and Engin Kirda. 2022. SoK: All or Nothing-A Postmortem of Solutions to the Third-Party Script Inclusion Permission Model and a Path Forward. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 206–222.
[49] Marius Steffens, Marius Musch, Martin Johns, and Ben Stock. 2021. Who's hosting the block party? studying third-party blockage of csp and sri. In *NDSS*.
[50] Marius Steffens, Christian Rossow, Martin Johns, and Ben Stock. 2019. Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild.. In *NDSS*. https://doi.org/ndss-paper/dont-trust-the-locals-investigating-the-prevalence-of-persistent-client-side-cross-site-scripting-in-the-wild/
[51] Marius Steffens and Ben Stock. 2020. PMForce: Systematically Analyzing postMessage Handlers at Scale.. In *ACM CCS*. https://doi.org/10.1145/3372297.3417267
[52] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. 2017. How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security.. In *USENIX Security Symposium*.
[53] Ben Stock, Stephan Pfistner, Bernd Kaiser, Sebastian Lekies, and Martin Johns. 2015. From Facepalm to Brain Bender: Exploring Client-Side Cross-Site Scripting.. In *ACM CCS*. https://doi.org/10.1145/2810103.2813625
[54] Mike Ter Louw, Karthik Thotta Ganesh, and VN Venkatakrishnan. 2010. AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements.. In *USENIX Security*.
[55] Michael Toth, Nataliia Bielova, and Vincent Roca. 2022. On dark patterns and manipulation of website publishers by CMPs. In *PETS 2022-22nd Privacy Enhancing Technologies Symposium*.
[56] UniConsent. 2022. Google Ad Manager Integration. Online https://www.uniconsent.com/docs/tutorials/gam-integration.
[57] UniConsent. 2022. Google global site tag Integration. Online https://www.uniconsent.com/docs/tutorials/gtag-integration.
[58] Steven Van Acker, Philippe De Ryck, Lieven Desmet, Frank Piessens, and Wouter Joosen. 2011. WebJail: least-privilege integration of third-party components in web mashups. In *ACSAC*.
[59] W3C Privacy Community Group. 2021. First-Party Sets. Online https://github.com/privacycg/first-party-sets.
[60] WIRED. 1999. Sun on Privacy: 'Get Over It'. https://www.wired.com/1999/01/sun-on-privacy-get-over-it/. Accessed 17.06.2022.