A preliminary study on the adoption and effectiveness of `SameSite` cookies as a CSRF defence*

Luca Compagna
SAP Labs France
firstname.lastname@sap.com

Hugo Jonker
Radboud University
Open University Netherlands
firstname.lastname@ou.nl

Johannes Krochewski
SAP Labs France
firstname.lastname@sap.com

Benjamin Krumnow
TH Köln
Open University Netherlands
firstname.lastname@th-koeln.de

Merve Sahin
SAP Labs France
firstname.lastname@sap.com

Abstract—**The `SameSite` cookie attribute was introduced to prevent Cross-site Request Forgery (CSRF) attacks. Major browsers support `SameSite` functionality since 2016. Since 2020, browsers enforce it by default. These developments sometimes have been celebrated as the end of CSRF. In this paper, we have a closer look into the potential of `SameSite` mechanism to effectively fight CSRF in practice. Our measurements and evaluations over most popular websites indicate that, if properly deployed, the `SameSite` mechanism can be effective against the major CSRF attack scenario. Still, like any other countermeasure, it is not likely to be a silver bullet to end CSRF, due to various scenarios that require additional protection. We refactor our findings in a set of guidelines for the web community on how to make best use of `SameSite` and what it is left to do to fight CSRF.**

Index Terms—**Browser security, CSRF, SameSite**

1. Introduction

Cross Site Request Forgery (CSRF) has been one of the most common web application vulnerabilities that has been around for almost 20 years [38], and stayed in the OWASP Top 10 list until the last release in 2017 release [27], when it was removed because ”More frameworks offering secure-by-default settings and some form of protections”.

Indeed various defence mechanisms are available, including anti-CSRF tokens, extra steps requiring user to provide explicit consent via re-authentication or captcha, and a very recent mechanism based on the `SameSite` cookie attribute that allows to specify if a cookie can be sent along with cross-site requests. (Throughout the paper, we will call this as the Same-Site Cookie mechanism, or SSC in short.) While these defences helped to demote and take out CSRF from the OWASP Top 10 list, recent studies show that CSRF attacks are still significant [4], [28], [36], [19]. In particular, CSRF can come in many forms (depending on authentication status of the user and/or on the scenario being executed) that may require the usage and combination of different defences in the same website.

For instance, a website may handle user account operations in-house and outsource payment checkout operations to third-parties like PayPal. All these operations need to be protected against CSRF, but while the in-house ones would trigger same-site requests (i.e., requests that originate from a website's domain, to be processed by the same domain itself), the payment ones will trigger cross-site requests in the context of a protocol with the third-party. While CSRF protection for the same-site requests are usually achieved by means of tokens bound to the user cookies, the session binding for all the cross-site requests require ad-hoc protection.

In this paper, we focus on the recent SSC mechanism and evaluate its impact on CSRF. SSC is an IETF standard proposal made in 2016 [41], which aims to prevent certain cookies from being attached to *cross-site* requests. Multiple blogs and posts have been published since the SSC proposal, mainly to welcome this feature as a key defence against CSRF [20], [26], [30]. Although it has been already few years since SSC was proposed, the adoption of the idea by the Chrome browser in 2020 brought it back to light: As of February 2020, Chrome 80 started to enforce SSC by setting any empty `SameSite` cookie attribute to `lax` by default [29]. (We will refer to this as the *LaxByDefault* policy throught the paper). In some cases, this has been celebrated and advertised as *the death of CSRF* [16].

But, does this celebration stands on solid ground? Is CSRF really dead thanks to the current SSC developments? Are websites embracing SSC or are they turning it down? Does it still make sense to invest in devising accurate testing approaches for CSRF? In this paper, we aim to answer these questions via the following contributions:

- **(C1)** a measurement of the most popular websites to evaluate their adoption of SSC,
- **(C2)** analysing the potential of SSC to prevent different types of CSRF vulnerabilities,
- **(C3)** evaluating several websites for CSRF vulnerabilities with respect to their SSC adoption, and
- **(C4)** presenting guidelines for the web community to make the best use of SSC.

Authors are listed in alphabetical order.

For (C1), we measure adoption of the `SameSite` cookie attribute from January 2020 to March 2020, during the transition period that Chrome started to enforce the *LaxByDefault* policy. We find that most websites continue ignoring SSC, and some of them partially or completely disable it. Even for authentication cookies, which are clearly security sensitive, we see a similar picture. This shows that browsers' enforcement of *LaxByDefault* policy is critical to benefit from SSC, and it impacts a significant ratio of cookies and websites.

In (C2), we analyse the ratio of state-changing requests covered by this policy, using two different datasets, one of which was borrowed from [4]. We find that around 14% of such requests are implemented via the `GET` method, which undermines the *LaxByDefault* protection.

We then further look into websites that intentionally disable *LaxByDefault* for their authentication cookies in (C3). We manually test 20 such websites, which we identified during our study of SSC adoption. We found several CSRF vulnerabilities on 9 of these sites. These were responsibly disclosed. All these findings show that, if properly deployed, SSC with *LaxByDefault* policy can be effective against the major CSRF attack scenario. Still, like any other countermeasure, it is not likely to be a silver bullet to end CSRF.

Finally in (C4), we propose several guidelines for web browsers, developers, and security testers to maximise SSC protection.

2. Background

2.1. Cross-site request forgery

Cross-site requests happen when a site makes the visitor's browser send an HTTP request to another site. Examples of benign uses are Single Sign-On (SSO) and third-party payment providers (e.g., Paypal, Stripe, etc.). *Cross-site request forgery* (CSRF) is an attack where the attacker triggers the victim browser to send an HTTP request to a website of the attacker's choosing. CSRF attacks can be used e.g., for tracking, but also for using functionality on other websites if the victim is logged in there.

From here on, we refer to requests that are not cross-site as *same-site* requests.

2.1.1. Pre-authentication vs post-authentication CSRF. CSRF attacks can be classified as either pre- or post-authentication, depending on whether the spoofed request requires the victim to have an already-established authenticated session or not [36]. Post-authentication CSRF is known at least since 2001 [38] and has received a lot of attention from the web community. Pre-authentication CSRF is known since 2008 [2], when its first instance, referred to as Login CSRF, was introduced.

In post-authentication CSRF, the attacker has the victim executing actions on the victim's authenticated session. In pre-authentication CSRF, the attacker tricks the victim into establishing an authenticated session that is under the control of the attacker. Depending on how the used site stores user interaction, the attacker can later see these interactions. For example: search engines typically allow users to log in to keep track of search history, sites supporting payment may allow a user to add a credit card (which is then added to the attacker's account), etc.

2.1.2. Client-side CSRF. Standard CSRF tricks the victim's browser into sending an HTTP request by sending it specially crafted HTML. Many websites make use of client-side JavaScript functions that make regular HTTP requests in the background. Recently, Khodayari et al. [19] described a new twist on CSRF attacks leveraging such client-side JavaScript functions. In particular, the attacker manipulates the input to these functions, causing them to send the attacker's desired HTTP request instead of the intended request. Note that existing CSRF protections, such as anti-CSRF tokens, operate at a lower level than JavaScript. Any JavaScript function that makes HTTP requests thus has to take existing anti-CSRF mechanisms on the leveraged website into account already. As such, leveraging a website's client-side HTTP request functions to execute a CSRF attack will bypass that site's CSRF defences, making this a particular insidious form of a CSRF attack.

2.2. Same-Site Cookies (SSC)

The `SameSite` cookie attribute was introduced as an Internet draft specification in 2016 [41]. The idea of the attribute is to allow the website to specify under which conditions a cookie can be sent along with cross-site requests. This would help protect against CSRF. The attribute has three options [8]:

- `strict`: The cookie is only sent along with same-site requests. Thus, for instance, the initial requests to the website would not have this cookie appended.
- `lax`: The cookie is sent along with same-site requests. It is sent with cross-site requests if and only if the request is a top-level navigation, and it uses a 'safe' HTTP method. These are defined in RFC7231: `GET`, `HEAD`, `OPTIONS`, and `TRACE`.
- `none`: The cookie is sent along with both same-site and cross-site requests. This option basically disables the SSC defence.

2.2.1. Default treatment for cookies lacking the `SameSite` attribute. Initially, if the same-site attribute is left blank, or contains an invalid value, browsers were to treat such cookies as if they are set to `SameSite=none`. This ensures that the original behaviour of the website will not be affected by the introduction of `SameSite`. However, this setting meant that by default, the new protection went unused. In 2019, a followup draft specification [39] proposed that browsers should instead treat absence of the `SameSite` attribute as `SameSite=lax`. In a further refinement in 2020 [40], cookies with `SameSite=none` were additionally required to use the `Secure` flag. Both Firefox [9] and Google [14] announced they would incorporate both changes. We refer to this new proposed browser policy (including rejecting `SameSite=none` cookies without a `Secure` flag) as *LaxByDefault* to differentiate it from the initial *NoneByDefault*. *LaxByDefault* makes `SameSite` CSRF protection the default for all cookies. Note that this comes with a cost: cookies without a specific `SameSite` attribute will not be available in a third-party context, potentially affecting current functionality.

2.2.2. Timeline of adoption by Chrome. Google started rolling out *LaxByDefault* for its Chrome browser on February 17, 2020 [15], [29]. The initial limited audience was slowly increased until April 3rd, where the experimental roll-out was halted due to the Covid-19 pandemic. On July 14th, the roll-out restarted, and on August 11th it reached 100% of target users (using Chrome 80 and above). By Chrome 85 (August 25th), this was the default setting for all users of Chrome. This was also incorporated into Chromium, the open-source project on which Chrome and other browsers (incl. Microsoft's Edge, Opera). These browsers started rollout with Chrome version 82, becoming default for all users later. For Microsoft Edge, this happened with version 86, released on October 9th.

3. Adoption of SameSite as defence

In this section, we provide a description of our data collection process and report about the datasets we created to measure the adoption of SSC. Within the analysis, we look at the adoption over time as well as the transition period. Finally, we zoom in on SSC adoption for security-sensitive cookies.

3.1. Data collection & categorisation

We measure the adoption of SSC based on cookies that are served by websites with respect to two aspects: the overall adoption of this security measure and the reaction to the switch. While the switch affects all cookies equally, not all cookies are of equal importance to a website's core functionality. Some cookies are security-sensitive (e.g., session cookies or authentication cookies)[1], whereas others are less so (e.g., language settings). Some security-sensitive cookies may only be served in the process of security-sensitive operations, such as logging in. We are interested in both types, wherefore we tailor our data acquisition accordingly. In detail, we conduct two different measurements:

1. Acquiring pre-login data: visiting the landing page to retrieve generic cookies, including session cookies for unauthenticated users.
2. Acquiring post-login data: logging in to retrieve the authentication cookies.

The first measurement provides a straight-forward approach to acquire cookies. As this can be easily repeated, we use pre-login scans to gain a finer resolution of websites during the switch to *LaxByDefault*. In contrast, acquiring post-login data is more complex; it requires the acquisition of credentials and to deep scan sites for login areas. Hence, we followed a different study design within the two data acquisitions. We provide a further description for each individual approach below.

1. Note that session cookies can be distinct from authentication cookies: session cookies keep track of the visitor's session, while authentication cookies prove the identity of the user. E.g., having sessions for unauthenticated visitors is possible.

3.1.1. Acquisition of pre-login data. To periodically acquire pre-login data from websites, we automate a Chrome browser via Selenium and WebDriver. We use Chrome v.79, which uses the *NoneByDefault* policy. This means cookies will be not automatically set to `SameSite=lax` by our client. To capture cookies, we setup the OWASP ZAP proxy[2] that stores all HTTP requests and responses.

We conduct seven runs to collect data of the Alexa Top 5K over a period of more than 2 months, starting 15^{th} of January 2020 (one month before Chrome's switch). To account for developments since then, we conducted another run once in April 2021 (15 months after the first scan). Each run consists of two visits of a single site to account for variations in served cookies by a site. The two resulting cookie sets are then merged into one set and already existing cookies are discarded. During each visit, we employ the selenium driver within our python framework to fully load sites under a timeout of 300 seconds. All measurements were run from a company network. For this work, we consider only the data from websites that we successfully visited in our seven runs. This leaves us with data from 3,534 websites.

3.1.2. Acquisition of post-login data. To gather data, we use Shepherd [17], a tool that can automatically log in on websites to perform data acquisition in post-login parts of sites. For that, Shepherd requires a database with credentials, which we acquired for 56,437 websites by feeding domains from the Tranco list [31] into BugMeNot[3]; a service offering crowed-sourced credentials. Note that BugMeNot's crowd-sourced data is not verified. The authors of Shepherd [17] found roughly 60% of credentials will be invalid in their work. Further other failure modes (unreachable sites, CAPTCHA, etc.) additionally lower the success rate. In our data collection, Shepherd successfully collected data from 6,180 websites. Our data acquisition happened in two runs, between April 3, 2020 and May 18, 2020.

Finally, BugMeNot contains credentials throughout the entire Alexa Top 1M websites. Still, as a crowd-sourced resource, it is dependent on user input, which is expected to favour more popular sites. Indeed, in a previous study using Shepherd [17], a large portion (79%) of the harvested credentials belonged to sites with ranks in the top 500K. Our dataset shows with 75% a similar distribution.

3.1.3. Selecting security-sensitive cookies. To separate security-sensitive cookies from other cookies, we use an individual method for the pre- and post-login dataset. For the post-login dataset, we let Shepherd automatically determine which cookies are authentication cookies. For that, Shepherd tests after logging-in whether the removal of a cookie breaks a session. However, the iterations needed to find all possible combinations of authentication cookies is known to grow exponentially [5], wherefore Shepherd relies on the algorithms proposed by Calzavara et al. [6] and Mundada et al. [25] to improve runtime performance.

In contrast, when running our pre-login data collection, we cannot leverage an authenticated session to

2. https://www.zaproxy.org/
3. http://bugmenot.com

deduct session cookies. Thus, we fallback to heuristics to select session cookies as proposed in earlier studies [37], [12]. Specifically, we interpret a cookie as session cookie, when (i) the cookie name contains a known case insensitive session cookie string, such as `sess, sessid, session, cfid, cftoken` and (ii) the cookie value exceeds either the entropy of 3.45, or the length of 69 characters (cf., [37]).

3.1.4. Categorisation of sites. Each cookie plays a different role in a website's functionality. Therefore, a website need not handle all cookies equally. For example, authentication typically relies on cookies, and thus a website's authentication process may not work in browsers using *LaxByDefault*. A website will need to address this, e.g. by setting all authentication cookies to `SameSite=none`, while leaving other cookies unchanged. Alternatively they may choose to set all their cookies to `none` to simplify the update, or may choose to update their authentication process to incorporate SSC, etc. Therefore, we categorise websites according to how they handle the `SameSite` attribute across their cookies:

- *Ignoring*: These websites never set any value for the `SameSite` attribute of their cookies, leaving it to the default behaviour of the browser.
- *Partially disabling*: These websites set some of their cookies' `SameSite` attribute to `none`, partially disabling the browser provided defence. They never use `lax` or `strict`.
- *Completely disabling*: These websites set all their cookies' `SameSite` attribute to `none`, completely disabling the browser provided defence.
- *Partially adopting*: These websites set some of their cookies' `SameSite` attribute to `lax` or `strict`, partially adopting the samesite defence. They never use `none`.
- *Fully adopting*: These websites set all their cookies' `SameSite` attribute `lax` or `strict`, fully adopting the samesite defence even when it is not enforced by the browser.
- *Mixed strategy*: These websites use both adoption related values (`strict` or `lax`), as well as the disabling feature `none`.

Note that the categorisation captures the intention of websites with respect to the `SameSite` attribute, regardless of whether they were able to correctly implement this intended behaviour. For instance, a cookie that has `SameSite=none` value may not have the `Secure` flag set correctly. This means that the cookie will not be sent along with third-party requests. We ignore such contradictory cases in our categorisation. We will report more on this in Section 3.2.4. Finally, it is important to keep in mind that the categorisation of any individual website can change over time, as it changes the `SameSite` attribute for its cookies.

3.1.5. Limitations. Note that there are some limitations stemming either from our methodology or the execution of the experiments.

First, either datasets will fail to cover all types of websites. Due to our company firewall, we assume that sites falling into categories like entertainment, adult content, or malicious sites will be underrepresented in the pre-login dataset. Our post-login dataset was not collected from behind the same firewall. Nevertheless, it is also not complete, due to its crowd-sourced nature. Credentials for sites that are valuable to users are unlikely to be shared (e.g., banks). Moreover, BugMeNot prohibits sharing of accounts that allow access to paid content or to circumvent age restrictions.

Finally, our selection of security-sensitive cookies for the pre-login dataset relies on heuristics. An evaluations by Calzavara et al. [5] has shown that heuristics are fairly limited in correctly identifying session cookies.

3.2. Adoption on pre-authentication cookies

3.2.1. The transition period. We start by looking at websites' reaction to the new *LaxByDefault* policy announcement by Chrome for the time between January 15 and March 30, 2020. Figure 1 shows the adoption on (a) first-party and (b) third-party cookies (collected from 3,534 websites), respectively.

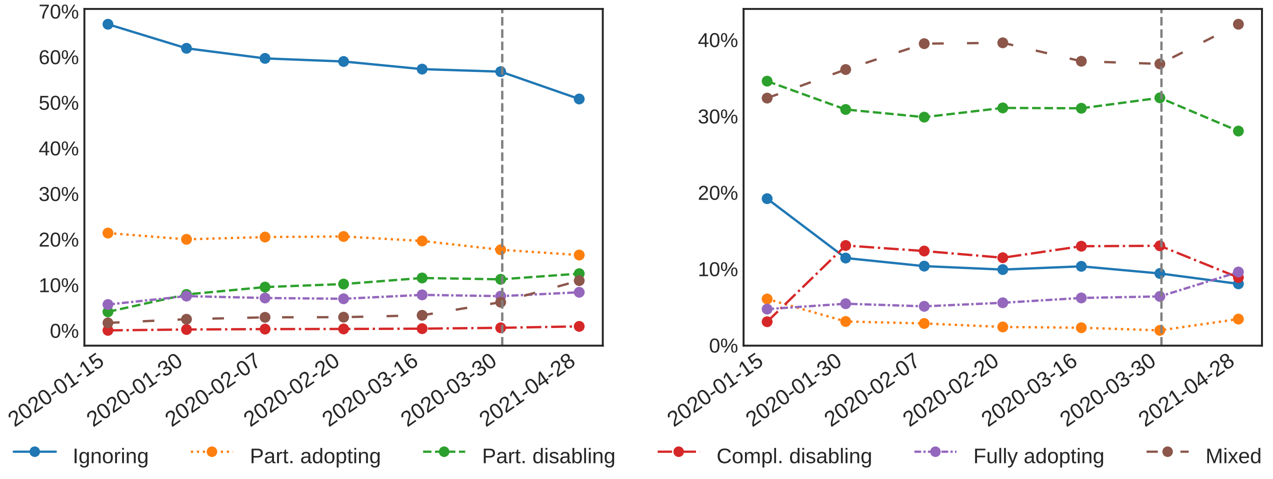
For the first-party cookies, applying a two-tailed z-test for two population proportions, we measure a significant decrease on websites ignoring the `SameSite` attribute (z-score = 9.0021, p-value < 0.00001) and a significant increase in websites that are partially disabling `SameSite` (z-score = 27.9684, $p < 0.00001$). Moreover, we observe a slight increase in the mixed strategy (4.5%).

For third-party cookies, we see a similar picture as above. The number of websites ignoring the `SameSite` attribute significantly drops by 10% (z-score = 12.1145, p-value < 0.00001) and sites following a mixed strategy increase by 4.5%. However, this time we find an increase in websites that completely – instead of partially – disable SSC (z-score = -15.4946, p-value < 0.00001). This makes sense, considering that the third-party cookies often relate to analytic, tracking or advertisement purposes, and thus websites need to send these in cross-site requests.

Overall, we observe that only around 10% of websites reacted to the *LaxByDefault* switch by changing their `SameSite` adoption strategy, and the most common reaction was to partially disable it for first-party cookies, and to completely disable it for third-party cookies.

By considering cookies instead of sites, we can gain an estimate of the cookie ratio that would be automatically upgraded by the *LaxByDefault* enforcement. In January 2020, we count that 96% of first-party cookies and 52% of third-party cookies leave the SameSite value undeclared, which would later result in upgraded to `lax`. This coverage decreases to 92% for first-party cookies and to 28% for third-party cookies over the transition period. However, when we include cookies that were upgraded directly by sites, e.g., by setting `SameSite` to `lax` or `strict`, we see an almost equal protection as at the beginning of our measurements (96% of first-party cookies and 45% of third-party cookies).

3.2.2. Session cookies first-party cookies. Next, we take a closer look at the first-party cookies that are likely to be session cookies. We base this analysis on the assumption that these cookies are used for the session management within the post-authentication phase, and are therefore security-sensitive.



(a) (b)

Figure 1. Timeline of `SameSite` adoption on (a) first-party cookies and (b) third-party cookies with respect to the ratio of websites. Note that there is a 13-months gap between the last two data points on the x-axis (i.e., 2020-0-30 to 2021-04-28), depicted by the vertical grey line.

Based on selection criteria described in Section 3.1.3, we identify candidates for session cookies on 1,480 websites. Among all scans, on average, a website has 1.9 ± 1.4 cookies that are likely to carry session identifiers. Figure 2 shows the `SameSite` adoption rate on these cookies. We can see that, while in January 2020 98% of websites were ignoring `SameSite`, this ratio dropped to 90% by the end of March 2020. The most common reactions were either to completely disable (7.4% increase), or to fully adopt (5.1% increase) `SameSite`. Thus, the 7.4% of the websites that completely disable the SSC defence might be susceptible to security issues.

3.2.3. Overall adoption over time. We contrast our data from the switch period with a new scan from April 2021 (13 months later) to gain a long term perspective. Our comparison shows a clear trend in the adoption of SSC.

For both, first-party and third-party cookies (see Figure 1), websites ignoring SSC have further decreased within the last 13 months, but slower than during the transition period. Overall, 51% of websites directly benefit from Chrome's *LaxByDefault* enforcement for the first-party cookies. Among the rest of the websites, the most common strategies are partial adoption and partial disabling.

3.2.4. Implementation issues with respect to `SameSite=none`. Here, we focus on the cookies with `SameSite=none` value, for which the `Secure` flag is not properly set. It is possible that some websites initially fail to properly implement this behaviour. However, as their cookies will be rejected, they will need to fix it to achieve their intended behaviour. Figure 3 shows the timeline of the ratio of websites that have improperly configured `SameSite=none` cookie, both for first and third party cookies.

Considering the first party cookies, the ratio of websites remain more or less constant, fluctuating between 0.7% and 3%. We assume that these websites do not experience a significant breakage due to this.

Considering the third party cookies, we observe that initially almost half of the websites we analyze had misconfigured third party cookies with respect to `SameSite=none`. However, these were rapidly fixed, and by March 2020, only 3% of websites are affected by this. We can again assume that such misconfigured third party cookies do not cause a significant problem in functionality.

3.3. Adoption on authentication cookies

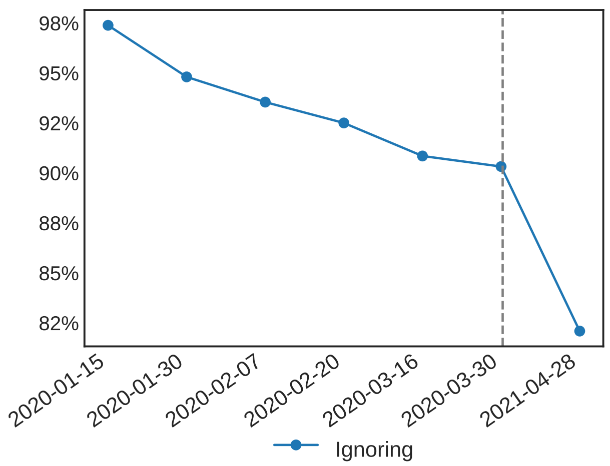
We now look at our dataset that we retrieved from post-login areas to investigate the use of SSC for authentication cookies. Table 1 shows the ratio of websites in each category of `SameSite` adoption. We observe that most of the websites do not take any action on their authentication cookies, while only a small portion (~4%) protects all their authentication cookies either by using `lax` or `strict`. Moreover, around 2.8% of the websites disable SSC completely. These websites offer an interesting vantage point to study, as it is unclear if they use other sorts of CSRF protections while disabling `SameSite`. We will have a closer look into this in the next section.

TABLE 1. WEBSITES' USAGE OF THE SAMESITE ATTRIBUTE ON AUTHENTICATION COOKIES.

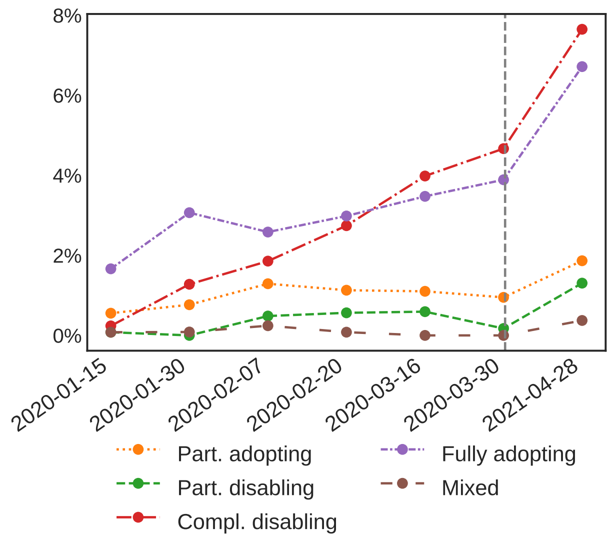
	Number of websites	
Total	**6,180**	**100.0%**
Ignoring	5,704	92.3%
Partially disabling.	35	0.5%
Completely disabling	116	1.9%
Partially adopting	55	0.9%
Fully adopting	259	4.2%
Mixed strategy	11	0.2%

4. Impact on CSRF

In this section, we follow two goals. First, we aim to explore the potential of SSC to protect websites against



(a)



(b)

Figure 2. Timeline of `SameSite` adoption on first-party cookies that relate to pre-authentication session IDs: (a) shows the Ignoring strategy, (b) shows the rest. Note that there is a 13-months gap between the last two data points on the x-axis.

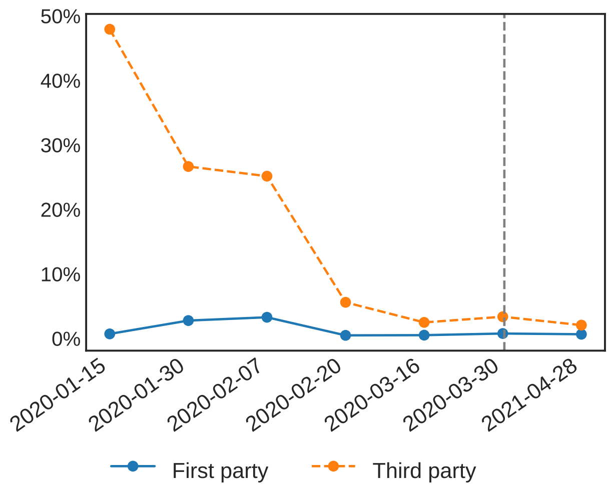


Figure 3. Timeline of `SameSite=none` misimplementation: showing the ratio of websites that have at least one `SameSite=none` cookie, without setting the `Secure` flag. Note that there is a 13-months gap between the last two data points on the x-axis.

CSRF under real-world conditions. Second, we investigate if there are specific reasons for websites to completely disable SSC, and what are the possible consequences of this in terms of CSRF vulnerabilities.

For the first goal, we need to identify the state-changing requests, as these are the potential targets for an attacker to conduct CSRF attacks. However, identifying state-changing requests is a non-trivial task. First, an evaluator should explore the website to discover sensitive functionality that an attacker could potentially exploit. Next, the evaluator must interact with the website to trigger the sensitive functionality and identify the corresponding state-changing request among various other requests. As this process does not scale with the number of sites we considered during our measurements, we use additional datasets where state-changing requests were already flagged.

For the second goal, we use our findings from Section 3.3, where we identified the websites that set all their post-login authentication cookies to `none` (i.e, completely disabling `SameSite` defence). We used these websites for a preliminary evaluation on vulnerability to CSRF.

4.1. Potential of SSC defence

We now analyse if state-changing requests, as occurring in the wild, would be potentially protected by SSC.

As mentioned earlier, we base our analysis on existing datasets to reach broader coverage. In particular, we use the following two datasets: Dataset **D-I** is publicly available and was created by Calzavara et al. [4] in 2017. To build this set, the authors manually traversed top ranked websites that provide a login. They further tagged (post-authenticated) state-changing requests among other requests as they used it to train a classifier. Our copy of this dataset[4] contains 6,204 HTTP requests from 51 websites of which 924 requests were marked as state-changing. The second set **D-II** is a manual collection that we created during earlier experiments in 2019[5]. Here, we also take pre-authentication scenarios into account (not considered in D-I). To build this set, we analysed a set of 47 websites, which consist of major e-commerce sites and sites belonging to common categories, such as entertainment software, news, etc. For 24 of these sites, we focused on the pre-authentication phase collecting HTTP requests related to login scenarios (some based on single sign-on), while the remaining 23 were analysed during the post-authentication phase where we collected HTTP requests related to a variety of operations including creation of a new invoice, the addition of a new administrator, data deletion, etc. More than 2000 HTTP requests were gathered and manually inspected by us to flag those associated

4. Our copy was extracted from https://github.com/alviser/mitch.
5. The dataset is available at https://github.com/compaluca/scr.

TABLE 2. POTENTIAL PROTECTION ON POST-AUTHENTICATION STATE-CHANGING REQUESTS

Dataset	request	GET	POST	DELETE / PUT	SSC potential
D-I	same-site	112	758	36	88%
	cross-site	9	7	2	0%
potential protection over D-I (6,204 requests over 51 websites): 86%					
D-II	same-site	5	52	4	92%
	cross-site	0	6	0	0%
potential protection over D-II (1,665 requests over 23 websites): 78%					
Combined		**126**	**823**	**42**	**86%**

to state-changing actions. In total, set D-II includes 67 HTTP requests flagged as state-changing requests.

In the below analysis of D-I and D-II, we differentiate between potential state-changing actions executed before and after authentication.

4.1.1. Pre-authentication (D-II).

For the 24 websites used for pre-authentication scenarios, we only focused on login related operations either based on standard form-based login (11 websites) or single sign-on (SSO) with an external identity provider (13 websites).

The latter category is clearly cross-site and it requires thus ad-hoc defences going beyond SSC and depending on the specific cross-site scenario. In our case, all tests targeted SSO scenarios based on the OAuth2 protocol whose protection against CSRF relies mainly on ensuring the `state` parameter is properly used to ensure session binding between different protocol steps.

The 11 websites featuring form-based login scenarios all used a simple POST request to send the user credentials. It shall be noticed that these scenarios can be protected via SSC defence. For instance, the website could simply use the session ID cookie that is set even before authentication. If this cookie is not having `samesite=none` and the website rejects login requests not having that session ID cookie, then the login operation cannot be triggered from a cross-site request crafted by an attacker.

Besides login, pre-authentication scenarios target registration and account validation. The latest one normally amount to execute a cross-site scenario where the user is required to validate a URL (with an non-guessable token) received by email. This it thus out-of-scope for SSC. On the other hand, the SSC-based defence explained for form-login would work also for form-based registration scenarios.

4.1.2. Post-authentication (D-I and D-II).

Table 2 overviews the state-changing requests per HTTP method extracted from datasets D-I and D-II. We separated these further into same-site and cross-site requests. In total, we count 991 (794 for D-I and 56 for D-II) state-changing requests, most of which are non-`GET` same-site requests (∼86%). These are the kind of state-changing requests where SSC has the potential to prevent CSRF-attacks. A web server with cookie-based sessions[6] will reject these requests if the authentication cookies are not transmitted due to SSC protection.

Considering the SSC adoption ratio from Section 3 and assuming that ratio can be transposed to the websites of datasets D-I and D-II, we can estimate:

- for users navigating with browsers using *LaxByDefault* (around 51.6% of all users, combining the figures from browser compatibility for SSC [24] with those on browser usage [34], see Appendix A for more details), SSC would be quite effective to protect most of those requests (∼84%)–86%*(100%-1.9%)– as only a minority of websites (1.9%, cf. Table 1) disable completely SSC over authentication cookies in same-site scenarios.
- for users navigating with browsers not using *LaxByDefault*, SSC protection would drop significantly to only ∼5% (∼86%*(100%-1.9%-92.3%)), as most of the websites (92.3%, cf. Table 1) ignore SSC over authentication cookies in same-site scenarios.

The remaining ∼14% relate with state-changing requests that are cross-site and/or executing the `GET` method (cf. Table 2). For these requests SSC is not effective to prevent CSRF. Let us dig a bit more into these requests.

- in general, `GET` requests should not be used to perform state-changing actions. We analysed the `GET` requests in dataset D-I and most of those requests would be better served by a `POST`. For instance, ∼30% of those `GET` requests relate with logout operations, that should rather be triggered via a `POST` as explained in many blogs and guidelines for web developers e.g., [33].
- for the remaining non-`GET` cross-site requests, indeed specific defences should be put in place to protect the state-changing actions from CSRF. In our small set of cross-site requests, they all relate to user account actions with a third-party identity provider.

4.2. Websites disabling SSC: risk of CSRF

From previous sections, it is clear that SSC has the potential to prevent CSRF for the vast majority of same-site scenarios. However SSC has been introduced only in 2016, almost 15 years after initial discoveries of CSRF, so that websites have most likely already implemented other defences against CSRF. This could be one reason why some websites may decide to ignore or even disable SSC. Another possible reason is when a website exposes an API to serve itself as well as other external websites, thus answering cross-site requests.

Considering the data collected for SSC adoption (cf. Section 3.3), we take a closer look into some of those websites that completely disable SSC, in order to understand (i) the reasons behind that choice and (ii) whether the CSRF risk was well considered.

Our dataset contains 116 such websites. We analyse the first 20 of them manually to see if they are susceptible to CSRF attacks or if they use other protection mechanism. For 9 of the websites (∼45%), we were able to

6. Note that some websites do not rely on cookies for user authentication. As we did not encountered such cases, we believe that this rather rare.

identify at least one CSRF vulnerability. These are mainly related to user account operations such as changing the user's basic information (7 cases), changing the email address (3 cases), deleting the account (1 case), and updating the password (1 case). Other cases relate to other kind of operations such as sending emails among website members, managing blogs and projects, etc. However, why these websites disable SSC remains a difficult question to answer. For 3 websites we noticed that they expose some APIs to handle user account operations, but for the other 17 websites we did not see any particular reason justifying the disabling of SSC.

5. Guidelines

We refactor the observations from previous sections to offer some practical guidelines for web browsers, developers, and testers in the web community. For users, options to address `SameSite` are limited. The main action they can take is to use an up-to-date browser, see Appendix A.

5.1. Web browsers

From a security point of view there is no doubt that web browsers should support SSC and enforce *LaxByDefault* as default policy. However we miss in this respect data about other dimensions and mainly about the impact of this default policy over breakages in websites. It is very likely that browser vendors have some visibility on those data and it would be great to have them shared with the community.

5.2. Web developers

Web developers are invited to not disable SSC, over their authentication cookies in particular, unless they have a very strong reason to do so.

Rather than ignoring SSC (i.e., cookies are set without the `samesite` attribute) and leaving the default policy to browsers, web developers are encouraged to take an active role and define which values of `samesite` better suit their authentication cookies.

Web developers should also be aware that although SSC can contribute significantly in fighting CSRF, it is not effective in all scenarios. Assuming the website is adopting SSC properly, its effectiveness against CSRF is limited to same-site scenarios not using `GET` requests. We encourage web developers to inspect all their state-changing actions and perform the following:

- **Minimise `GET`**: If any of those state-changing actions involve `GET` requests, consider changing those requests into `POST`, `PUT`, etc. A classical example in this respect is the logout action that is often triggered via a `GET`, while a `POST` is recommended.
- **Protect cross-site scenarios**: Cross-site scenarios require specific CSRF protection that goes beyond SSC. Web developers are thus encouraged to check if they have cross-site scenarios in place and, in case, use the right defence among the ones suggested by the community for the specific cross-site scenario.
- **Beware of pre-authentication CSRF**: Pre-authentication state-changing scenarios are often overlooked by the web development community leaving many websites vulnerable to the variant of login CSRF. For those vulnerable websites, an attacker can authenticate victims under accounts owned by the attacker herself, enabling, e.g., the attacker to spoof victim activity in the website. The good news is that there are only few pre-authentication scenarios that needs to be protected, namely registration, account validation and login. We encourage web developers to follow standard guidelines to protect these few scenarios.
- **Beware of client-side CSRF**: Client-side CSRF is a quite recent variant of CSRF, first identified in 2018 [1]. The root cause for this variant is client-side JavaScript code executing asynchronous HTTP requests whose URL is not sanitised and can be manipulated by an attacker. Web developers are encouraged to identify those JavaScript code areas and to ensure they are properly protected.

5.3. Testers

Here we discuss how the testing strategy for CSRF can be made more effective by considering the recent SSC defence. Given a website, the following steps can be performed (ordered by priority):

Testing for post-authentication state-changing actions:

- Inspect authentication cookies (absence of one of these cookies makes the website lose the authenticated session) and check whether all of them are having `samesite=none`.
- If this is the case the website is opting-off SSC defence and thus the full-fledged standard testing strategy for CSRF should be applied (see next section for some experiment in this respect).
- Otherwise, the tester can save a lot of time by focusing only on (i) client-side CSRF, (ii) cross-site scenarios, and (iii) state-changing actions awkwardly triggered by `GET` HTTP requests.
- Indeed for all the other state-changing actions, the SSC defence will be sufficient to prevent CSRF.

From the data we analysed the majority of websites could get advantage of this simplified testing strategy, rechannelling the testing effort over

- state-changing actions using, for whatever reasons, the `GET` method. Our data comprises $\sim$12% of such state-changing requests;
- cross-site scenarios. Likely very few, according to our data ($\sim$2.4%);
- JavaScript code sending HTTP requests to mitigate the risk of client-side CSRF. We do not have figures in that respect from our data, but the study done in [19] provides scary numbers.

Testing for pre-authentication state-changing actions: While post-authentication state-changing actions rely almost by construction on authentication cookies, the same does not apply for pre-authentication ones, though nothing would prevent the web development community to go in that direction to take advantage of SSC defence in this context as well. Cross-site scenarios would still need to be tested apart, but same-site scenarios would be covered.

For the moment, however, the testing strategy for pre-authentication shall stay the same and the tester shall thus inspect that the registration, account validation and login operations are correctly implemented free of CSRF.

6. Related work

At the best of our knowledge, there is no other study in literature about SSC and its impact on CSRF. However, there is a extensive literature on both (i) CSRF and (ii) measuring security relevant HTTP attributes. Hereafter we discuss those.

6.1. CSRF related studies

CSRF is a long-standing issue, initially being mentioned in online forums around 2 decades ago [22]. There have been several defence mechanisms proposed by academic community since then. One of the first studies by Jovanovic et al. [18] proposes a server-side proxy to automatically add and validate CSRF tokens within requests. A later study aims to use heuristics to infer user intention (e.g., clicking a link, entering in address bar) and to remove the authentication tokens from the security sensitive requests in certain scenarios [22]. In a similar vein, De Ryck et al. [11] tries to infer if a website has explicitly delegated control to another website for certain cross-site scenarios. If such a "trusted delegation" does not exist, authentication tokens are stripped from the request. Another study [10] proposes a browser extension that allows to define and enforce a policy for cross-site requests, such as allowing, blocking or allowing without authentication token. Such browser enforced policy can be considered similar to the *LaxByDefault* policy.

On the other hand, some studies focus on automatically discovering the CSRF vulnerabilities in websites. For instance, Deemon [28] aims to identify state-changing requests via dynamic analysis and property graphs that include data flows and execution traces. These candidate requests are then tested for CSRF. A more recent study [4] aims to identify CSRF vulnerabilities with machine learning, using a set of manually labeled state-changing requests.

Finally, certain studies focus on the relatively unknown or new types of CSRF attacks. In particular, the client-side CSRF attacks have been recently introduced in [19]. In there the authors explicitly mentioned that SSC would not be effective against client-side CSRF. Moreover, [36] makes a large scale analysis of pre-authentication CSRF vulnerabilities.

6.2. Large-scale measurements on security relevant attributes

There has been various work studying cookie attributes, security relevant HTTP headers, and browser security features. For instance, Mendoza et al. [23] contrasted security-related fields in HTTP headers between mobile and desktop browsers and found inconsistencies between them. In contrast, Cahn et al. [3] conducted a large-scale study of cookies, collecting 3.2M cookies from Alexa Top 100K sites. They found that the amount of third-party cookies is two times larger than first-party cookies, and a small number of entities are able to aggregate user information across 75% of the web. Stock et al. [35] analysed the use of security headers for a 10 years period (2006-2016) based on the Internet Archive. Their study does not consider the `SameSite` attribute, as this defence was introduced later.

Franken et al. [13] evaluated browser implementations to secure third-party cookies with tracking protection mechanisms, but also with the `SameSite` attribute. They find that browsers (Chrome+Opera and Edge) can violate the `SameSite` policies (they send cookies, even it was lax or strict), due to faulty implementations.

Luo et al. [21] studied which security features are supported by mobile browsers. They found that some mobile browsers do not support `SameSite` functionality, leaving web users without protection. In addition, they scanned the landing pages of sites within the Alexa Top 50K and found that only 93 of them make use of the `SameSite` attribute.

Calzavara et al. [7] found inconsistencies emerging from the multiple client-enforced security policies deployed through HTTP headers. SSC is among the studied policies. Their data collection encompasses a single scan of 15,000 sites without logging in in Q1 2020. Thought their goal is not measuring adoption of SSC, their figures in that respect are in line with ours: Too few cookies (5% in their study) use SSC.

Finally, Sanchez-Rola et al. [32] studied timing attacks on cookies to expose web users' browsing history. In the course of their investigation, the authors measured the use of the `SameSite` cookie attribute as a defence against these attacks. As their analysis is tailored to one specific type of cookies, the overall adoption cannot be derived from their work. Nevertheless, the results of the presented subset provides some insights about the adoption in 2019: Settings like `none` and `strict` were rarely used, while sites that set the `SameSite` attribute typically used `lax` for a smaller portion of their cookies. The majority of sites left this option unset.

7. Conclusions

In this paper, we presented a preliminary study on the effectiveness of the recently introduced `SameSite` attribute and *LaxByDefault* policy to protect against CSRF in practice. We find that when browsers employ the *LaxByDefault* policy, SSC defence can be very effective, in particular in preventing CSRF against same-site state-changing actions that take place after authentication, and not triggered via the `GET` method. However, there are still certain CSRF variants that are not covered by SSC, and that should not be neglected. In particular, cross-site state-changing scenarios and client-side CSRF attack need to be addressed on their own, as SSC cannot prevent them. Finally, we provide a number of guidelines for the web community that, if implemented, would strengthen the fight against CSRF, taking also advantage of the SSC defence.

Future work. Given the preliminary nature of our investigation, we see various directions to expand our research. First, we plan to extend our evaluation for the impact on CSRF in the wild. For that, we consider exploring more specific cases that we identified during this work. In particular, we would like to investigate more websites that completely disable SSC and include websites that rely on third-party identity providers. Second, we plan to expand our study to include the time period before the beginning of our measurements. Therefore, we have already started to incorporate data from the Internet archive. Finally, SSC has the potential to break website functionality. Studying the different cases of breakage and how browser vendors handled this situation in general could provide useful insights for the introduction of new security features in the future.

Acknowledgements. We are grateful for the comments by the anonymous reviewers, which helped improve the paper significantly.

References

[1] Client-side CSRF at Facebook. https://www.facebook.com/notes/facebook-bug-bounty/client-side-csrf/2056804174333798/, 2018. Accessed July 15, 2021.

[2] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM*, CCS '08, pages 75–88, New York, NY, USA, 2008. ACM.

[3] Aaron Cahn, Scott Alfeld, Paul Barford, and S. Muthukrishnan. An empirical study of web cookies. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 891–901, Republic and Canton of Geneva, CHE, 2016.

[4] Stefano Calzavara, Mauro Conti, Riccardo Focardi, Alvise Rabitti, and Gabriele Tolomei. Mitch: A machine learning approach to the black-box detection of CSRF vulnerabilities. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 528–543. IEEE, 2019.

[5] Stefano Calzavara, Gabriele Tolomei, Michele Bugliesi, and Salvatore Orlando. Quite a mess in my cookie jar!: leveraging machine learning to protect web authentication. In *Proceedings of the 23rd International Conference on the World Wide Web, WWW'14*, pages 189–200. ACM, 2014.

[6] Stefano Calzavara, Gabriele Tolomei, Andrea Casini, Michele Bugliesi, and Salvatore Orlando. A supervised learning approach to protect client authentication on the web. *ACM Transactions on the Web, TWEB*, 9(3):15:1–15:30, 2015.

[7] Stefano Calzavara, Tobias Urban, Dennis Tatang, Marius Steffens, and Ben Stock. Reining in the web's inconsistencies with site policy. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021*, February 2021.

[8] Lily Chen, Steven Englehardt, Mike West, and John Wilander. Cookies: HTTP State Management Mechanism. Internet-Draft draft-ietf-httpbis-rfc6265bis-08, Internet Engineering Task Force, June 2021. https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis-08.

[9] Mike Conca. Changes to SameSite Cookie Behavior – A Call to Action for Web Developers. https://hacks.mozilla.org/2020/08/changes-to-samesite-cookie-behavior/, 2020. Accessed July 15, 2021.

[10] Philippe De Ryck, Lieven Desmet, Thomas Heyman, Frank Piessens, and Wouter Joosen. CsFire: Transparent client-side mitigation of malicious cross-domain requests. In *Engineering Secure Software and Systems, ESSoS*, pages 18–34, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] Philippe De Ryck, Lieven Desmet, Wouter Joosen, and Frank Piessens. Automatic and precise client-side protection against CSRF attacks. In *European Symposium on Research in Computer Security, ESORICS*, volume 6879, pages 100–116. Atluri, V, Springer, 2011.

[12] Philippe de Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, and Wouter Joosen. Serene: self-reliant client-side protection against session fixation. In *Proceedings of the 12th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 7272 of *LNCS*, pages 59–72. Springer, 2012.

[13] Gertjan Franken, Tom van Goethem, and Wouter Joosen. Who left open the cookie jar? A comprehensive evaluation of third-party cookie policies. In *USENIX Annual Technical Conference, USENIX ATC 2019*. USENIX Association, 2019.

[14] Google. Developers: Get Ready for New SameSite=None; Secure Cookie Settings. https://blog.chromium.org/2019/10/developers-get-ready-for-new.html, 2019. Accessed July 15, 2021.

[15] Google. Samesite update. https://www.chromium.org/updates/same-site, 2019, Accessed July 15, 2021.

[16] Scott Helme. CSRF is (really) dead. https://scotthelme.co.uk/csrf-is-really-dead/, 2019. Accessed July 15, 2021.

[17] Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and Marc Sleegers. Shepherd: a generic approach to automating website login. In *Proceedings of the 2nd Workshop on Measurements, Attacks and Defenses for the Web (MADWEB'20)*, pages 1–10. IEEE, 2020.

[18] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing cross site request forgery attacks. In *SecureComm*, pages 1–10. IEEE, 2006.

[19] Soheil Khodayari and Giancarlo Pellegrino. Jaw: Studying client-side CSRF with hybrid property graphs and declarative traversals. In *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, B.C., August 2021. USENIX Association.

[20] Sjoerd Langkemper. Preventing CSRF with the same-site cookie attribute. https://www.sjoerdlangkemper.nl/2016/04/14/preventing-csrf-with-samesite-cookie-attribute/, 2016. Accessed July 15, 2021.

[21] Meng Luo, Pierre Laperdrix, Nima Honarmand, and Nick Nikiforakis. Time does not heal all wounds: A longitudinal analysis of security-mechanism support in mobile browsers. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019*. The Internet Society, 2019.

[22] Ziqing Mao, Ninghui Li, and Ian Molloy. Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In *Proceedings of the 10th Annual Information Security Symposium*, CERIAS '09. CERIAS - Purdue University, 2009.

[23] Abner Mendoza, Phakpoom Chinprutthiwong, and Guofei Gu. Uncovering HTTP header inconsistencies and the impact on desktop/mobile websites. In *Proceedings of the 27th International Conference on World Wide Web 2018*, WWW '18, page 247–256, Republic and Canton of Geneva, CHE, 2018.

[24] Mozilla. Browser compatibility. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite#browser_compatibility, 2021, Accessed July 15, 2021.

[25] Yogesh Mundada, Nick Feamster, and Balachander Krishnamurthy. Half-baked cookies: Hardening cookie-based authentication for the modern web. In *the 11th ACM Asia Conference on Computer and Communications Security, AsiaCCS*, pages 675–685. ACM, 2016.

[26] Netsparker. Using the Same-Site cookie attribute to prevent CSRF attacks. https://www.netsparker.com/blog/web-security/same-site-cookie-attribute-prevent-cross-site-request-forgery/, 2020. Accessed July 15, 2021.

[27] OWASP. Owasp Top Ten. https://owasp.org/www-project-top-ten/, 2017. Accessed July 15, 2021.

[28] Giancarlo Pellegrino, Martin Johns, Simon Koch, Michael Backes, and Christian Rossow. Deemon: Detecting CSRF with dynamic analysis and property graphs. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1757–1771. ACM, 2017.

[29] Chrome Platform. Cookies default to SameSite=Lax. https://www.chromestatus.com/feature/5088147346030592, 2020. Accessed July 15, 2021.

[30] Chrome Platform. Defending against CSRF with SameSite cookies. https://portswigger.net/web-security/csrf/samesite-cookies, 2021. Accessed July 15, 2021.

[31] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019*. The Internet Society, 2019.

[32] Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos. Cookies from the past: Timing server-side request processing code for history sniffing. *Digital Threats: Research and Practic*, 1(4), 2020.

[33] Jordan Simpson. Should Logging Out Be a GET or POST? https://www.baeldung.com/logout-get-vs-post, 2021, Accessed July 15, 2021.

[34] Stetic. Browser statistics with version. https://www.stetic.com/market-share/browser/, 2021, Accessed July 15, 2021.

[35] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. How the web tangled itself: Uncovering the history of client-side web (in)security. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 971–987, Vancouver, BC, August 2017. USENIX Association.

[36] Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli. Large-scale analysis & detection of authentication cross-site request forgeries. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 350–365. IEEE, 2017.

[37] Shuo Tang, Nathan Dautenhahn, and Samuel T. King. Fortifying web-based applications automatically. In *CCS*, pages 615–626. ACM, 2011.

[38] Peter Watkins. Mail about CSRF to the BugTraq mailing list. https://web.archive.org/web/20020204142607/http://www.tux.org/~peterw/csrf.txt, 2001. Accessed July 15, 2021, via The Internet Archive.

[39] Mike West. Incrementally Better Cookies. Internet-Draft draft-west-cookie-incrementalism-00, Internet Engineering Task Force, 2016. https://datatracker.ietf.org/doc/html/draft-west-cookie-incrementalism-00.

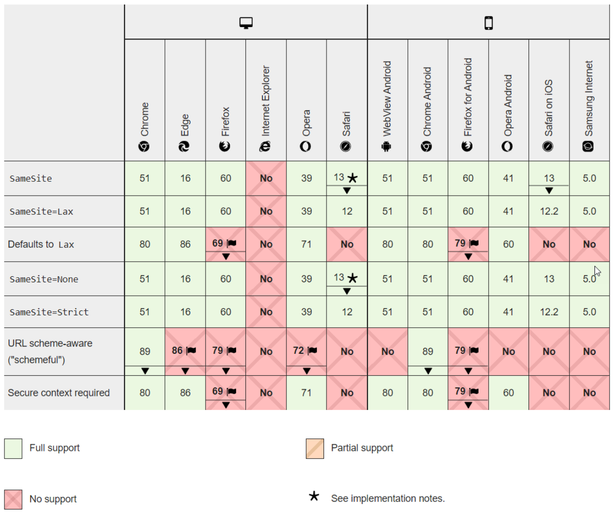
[40] Mike West. Incrementally better cookies draft-west-cookie-incrementalism-01. Technical report, Internet Engineering Task Force, 2020. https://datatracker.ietf.org/doc/html/draft-west-cookie-incrementalism-01#section-3.2.

[41] Mike West and Goodwin Mark. Same-site cookies – draft IETF 6265. Internet-draft, Internet Engineering Task Force, 2016. https://tools.ietf.org/pdf/draft-west-first-party-cookies-07.pdf.

Appendix

1. Browser: statistics and compatibility with SSC

In this section we report on few statistics about (i) browser compatibility with SSC (see Figure 4 borrowed from [24]) and (ii) market share of major browsers among users (see Figure 5 borrowed from [34]). Both the snapshots were captured on June 2020. By combining the two and focusing on web browsers only, we can observe that 51.6% of all users are employing browsers enforcing *LaxByDefault*. This percentage is computed by observing that only Chrome and Edge are currently enforcing the *LaxByDefault* policy.



	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
SameSite	51	16	60	No	39	13 ★	51	51	60	41	13	5.0
SameSite=Lax	51	16	60	No	39	12	51	51	60	41	12.2	5.0
Defaults to Lax	80	86	69	No	71	No	80	80	79	60	No	No
SameSite=None	51	16	60	No	39	13 ★	51	51	60	41	13	5.0
SameSite=Strict	51	16	60	No	39	12	51	51	60	41	12.2	5.0
URL scheme-aware ("schemeful")	89	86	79	No	72	No	No	89	79	No	No	No
Secure context required	80	86	69	No	71	No	80	80	79	60	No	No

Full support

Partial support

No support

★ See implementation notes.

Figure 4. Browser compatibility [24]

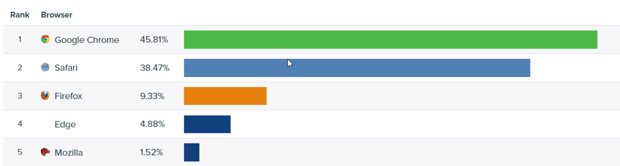


Figure 5. Browser statistics [34]