CSRF-ing the SSO waves: security testing of SSO-based account linking process

Andrea Bisegna
Center for Cybersecurity, Fondazione Bruno Kessler
Trento, Italy
a.bisegna@fbk.eu

Matteo Bitussi
Center for Cybersecurity, Fondazione Bruno Kessler
Trento, Italy
matteobitussi@gmail.com

Roberto Carbone
Center for Cybersecurity, Fondazione Bruno Kessler
Trento, Italy
carbone@fbk.eu

Luca Compagna
SAP Security Research
France
luca.compagna@sap.com

Silvio Ranise
Center for Cybersecurity, Fondazione Bruno Kessler and Department of Mathematics, University of Trento
Trento, Italy
ranise@fbk.eu

Avinash Sudhodanan
Independent Researcher
CA, USA
6.avinash@gmail.com

***Abstract*—The Single Sign-On based account linking process (SSOLinking in short) allows users to link their accounts at Service Provider (SP) websites to their Identity Providers (IdP) accounts. We focus on a serious (and overlooked) attack, namely an Account Hijack targeting the SSOLinking and relying on two CSRF vulnerabilities, one affecting the IdP and the other the SP. The former is an Authentication CSRF (also known as Login CSRF) and the latter is a CSRF on the button triggering the SSOLinking. We propose a security testing approach to help testers automatically detect such attacks. We implemented our testing technique as an extension (namely SSOLinking Checker) to the open-source penetration testing tool Micro-Id-Gym. To demonstrate the effectiveness of our approach and the pervasiveness of the SSOLinking Account Hijack, we conducted an experimental analysis against a selection of popular SPs that offer the SSOLinking with major IdPs. The results of our experiments are alarming: out of the 648 web sites we considered, 48 qualified for conducting our experiments and 21 of these suffered from SSOLinking vulnerability (i.e. 43.7%). Our findings (we responsibly disclosed to the affected vendors) include severe vulnerabilities among the web sites of Goodreads, Naver, Workable, etc.**

1. Introduction

More and more websites enrich their standard authentication processes with Single Sign-On (SSO) to smoothly sign-in users via popular Identity Providers (IdPs) like Facebook and Google. Ensuring the security of these SSO processes is thus paramount. A little mistake in their implementation may introduce a vulnerability and jeopardize the entire authentication, sometimes even enabling an attacker to take complete control of a victim's account on a website.

For instance, Cross-Site Request Forgery (CSRF) [25] is a vulnerability that enables a malicious website to forge state-changing HTTP requests from a victim's web browser. CSRF has been known for more than 20 years and has been demoted and removed from the OWASP Top 10 list in 2017 [26], mainly because of the rollout of framework supported defenses [17]. More recently, browser vendors rolled out *SameSite* cookies [24] and Fetch metadata headers [23] to further eradicate CSRF attacks. However, most of these defenses are built to ensure session binding within the workflows executed in a single website and may not be effective in SSO processes that are cross-site by construction. For instance, the recent SameSite cookies solution, enforced by major browsers like Chrome since 2020, aims to prevent certain cookies from being sent along with cross-site requests so to invalidate these requests. But SSO processes strongly rely on cross-site requests and thus such a defense, if not disabled for the specific SSO related cookies, would simply break the execution of the SSO processes. Protecting an SSO process against CSRF requires a careful combination of specific CSRF defenses built for the SSO process itself (e.g., the `state` parameter within the OAuth 2.0 protocol [21]) together with standard CSRF defenses preventing the SSO process from being unintentionally executed.

In this paper, we demonstrate that CSRF is far from being a solved problem for cross-site scenarios such as those implemented with SSO processes. On the contrary, we advocate the importance of *(i)* raising more awareness for CSRF issues in these scenarios as well as *(ii)* designing dynamic security testing techniques to support testers in detecting these issues. In our study, we focus on the SSO-based account linking process (*SSOLinking*, in short) and on an overlooked CSRF attack vector for SSO which was introduced by Rich Lundeen at the BlackHat conference in 2013 [22].

SSOLinking allows users to link their accounts at Service Provider (SP) websites, to their IdP account. This enables these users to authenticate at SPs through an IdP account, thereby eliminating the need to maintain a separate set of credentials for each SP. This added advantage encourages SPs to support the SSOLinking process. In fact, our experiments indicate that around 13% of the top 200 SP websites implementing the standard SSO login also support SSOLinking.

The CSRF attack vector we focus on in this paper relies on two different CSRF vulnerabilities, one affecting the IdP and the other the SP. The first vulnerability is an Authentication CSRF [34]—whose well-known instance is Login CSRF [5]—at the IdP. This vulnerability enables an attacker to authenticate a victim into an attacker-controlled account at the IdP. The second one is a CSRF on the action initiating the SSOLinking process at the SP (hereinafter referred to as *SSOLinking-Init-CSRF*). By combining both vulnerabilities, an attacker can craft an exploit web page that authenticates the victim to an attacker-controlled IdP account and initiates the SSOLinking process to connect the attacker's IdP account to the victim's SP account. Upon successful exploitation, the attacker will be able to authenticate to the victim's SP account by executing the SSO login at the SP through the attacker's IdP account. By doing so, the attack leads to a complete account hijacking, granting the attacker full control over the victim's SP account.

Our analysis shows that 43.7% of the SPs implementing the SSOLinking feature are vulnerable to SSOLinking-Init-CSRF. By combining the two vulnerabilities, an attacker can execute an account hijack of the victim at the SP. In this paper, as mentioned earlier, we will refer to this attack as *SSOLinking Account Hijack*. Although this attack was discussed by Lundeen *et al.* in 2013 [22], it was not considered in past research that assessed the implications of CSRF attacks on the SSOLinking process (e.g., [36], [19], [37], [34]). This may have led to the lack of awareness of this attack.

We inspect the two CSRF vulnerabilities and design a testing strategy to detect them. Considering that the number of IdPs is significantly less than that of SPs, that a handful of very popular IdPs is serving the majority of SPs [13], and that IdPs are quite reluctant to fix Authentication CSRF, we believe it is not worth to invest on implementing an automated testing technique for IdPs. We rather opted to test manually the four most popular IdPs identified in [13] against Authentication CSRF and found out that three of them are vulnerable.

Under the assumption that IdPs may be vulnerable to Authentication CSRF, we developed an automated testing technique for SSOLinking-Init-CSRF. The technique relies on existing technologies [8] that we extended to support our testing strategy. The goal is to empower a tester at the SP, the party that is impacted by the account hijack, with a security testing technique to detect the issue at the SP side, without requiring strong security expertise. Indeed, the tester shall only provide a Selenium script that executes the SSOLinking process with a specific IdP. Writing this kind of scripts is common practice for web applications as they act as simple integration tests, validating that the process executes as expected. Our security testing technique, which we implemented into the SSOLinking Checker prototype, runs and observes the execution of this script and performs additional automatically-generated tests in the background to check whether the process is vulnerable. When a vulnerability is detected, the SSOLinking Checker also generates an HTML Proof-of-Concept (HTML POC) webpage that allows the tester to easily reproduce the issue. This webpage is used both by SSOLinking Checker to perform the attack and as a proof for the SP to test its service.

For each major IdP, we select 12 popular SPs featuring the SSOLinking process and run SSOLinking Checker against them. The results are quite alarming: 21 out of 48 selected unique pairs SP-IdP are vulnerable to SSOLinking-Init-CSRF. We reported our findings to all the affected vendors. We are still interacting with some vendors to clarify the issues. Some of them already confirmed (and patched) the vulnerability we reported and we received some monetary rewards in the context of bug bounty programs.

These results confirm our conjecture concerning the lack of awareness and the need of security testing techniques, like those implemented in the SSOLinking Checker, to support testers.

To summarize, the contributions of this paper are as follows:

1) We propose an approach to assist a tester at the SP to automatically detect vulnerabilities enabling the SSOLinking Account Hijack. We have implemented our approach into the SSOLinking Checker prototype, based on the Burp Suite [12].
2) We demonstrate the effectiveness of our approach and the pervasiveness of the SSOLinking Account Hijack by performing an experimental analysis against a selection of popular websites that offer the SSO-based account linking process with major IdPs (48 unique pairs SP-IdP).
3) We detected and reported many CSRF vulnerabilities on prominent SPs and IdPs, enabling the SSOLinking Account Hijack:
   - 3 major IdPs (namely Google, Facebook and LinkedIn) are vulnerable to Authentication CSRF;
   - 21 unique pairs SP-IdP (out of 48 analyzed) are affected by SSOLinking-Init-CSRF, including Goodreads, Naver and Workable;
   - We responsibly disclosed our findings and received some monetary rewards.
4) We noticed that the large majority 19/21 of the vulnerable SPs had protected their SSOLinking flow from the popular CSRF attack involving the absence (or incorrect validation) of the OAuth 2.0 `state` parameter. This finding indicates the importance of studies like ours to spread awareness of the SSOLinking Account Hijack.

Structure of the paper. In Section 2, we introduce some background. In Section 3 we discuss the SSOLinking process and the SSOLinking Account Hijack. Then, in Section 4, we present our approach to support SP testers. In Section 5, we describe the experimental analysis on popular IdPs and SPs, by detailing the dataset selection procedure, the methodology we followed for the analysis and the results. In Section 6, we delve into various mitigations for the prevention and detection of the SSOLinking Account Hijack. In Section 7 and Section 8, we present the responsible disclosure process and some limitations of our work, respectively. Section 9 presents some related work. We conclude and overview future work in Section 10.

2. Background

This section provides background knowledge on various types of CSRF attacks, widely-used defenses to prevent them, the testing framework we used, and obstacles to test automation.

2.1. CSRF Attacks

The Cross-Site-Request-Forgery (CSRF) attack is also known as *sea surf* or *session riding* attacks, taking advantage of the inherent statelessness of the web to simulate user actions from one website to another. Typically, CSRF is used to perform actions on behalf of the attacker using the victim's authenticated session. If a victim is logged into a website, an attacker can compel the victim's browser to execute actions on the same site by sending a forged HTTP request. In the context of this work, our focus is on cross-sites, as it involves two entities. A successful CSRF attack can be detrimental to both businesses and users, potentially leading to unauthorized fund transfers, password changes, identity theft, data theft, and the theft of session cookies. In the past, CSRF attacks were thought to only affect state-changing actions caused by authenticated users due to the assumption that only authenticated users can perform high-impact actions, such as making purchases from one account to another.

In [5], Barth *et al.* introduced the concept of Login CSRF, where an attacker tricks a victim's browser into sending an HTTP request to the authentication endpoint of a website with the attacker's credentials. As a result, the victim gets authenticated as the attacker. In this scenario, the attacker can monitor the actions performed by the victim on the vulnerable website, allowing the attacker to steal sensitive information from the victim. The authors provided examples from Google and PayPal to illustrate the impact of this attack. More recently, several variants of the Login CSRF attack have been reported in the Single Sign-On (SSO) domain [9], [3].

In [34], the authors classified CSRF attacks into two categories: *(i)* Non-Authenticated CSRF attacks, which do not require the victim to have an authenticated session with the vulnerable website, and *(ii)* Authenticated CSRF attacks, which are variants that necessitate the victim to have a valid authenticated session.

2.2. Defenses for CSRF

While there are many ways to defend websites against CSRF attacks [20], four primary approaches stand out: token-based, fetch metadata-based, `SameSite` cookies, and user interaction-based.

Token-Based. It helps a web site to maintain session integrity by using a secret token. The token is a unique, non-guessable value, cryptographically bound to the session identifier (e.g., using the Set-Cookie HTTP header), that is generated by the server-side application and transmitted to the browser in such a way that it is included in all the HTTP request made by the browser. In case the HTTP request contains a not valid token, the server-side application rejects the request. The usage of CSRF tokens can prevent CSRF attacks by making it impossible for an attacker to build a fully valid HTTP request suitable for feeding to a victim user. Since the attacker cannot determine or predict the value of a user's CSRF token, the attacker cannot build a request with all the parameters that are necessary for the application. In the case of the SSO scenario and more specifically in OAuth 2.0, the usage of the `state` parameter if seeded with a secure random, should avoid CSRF attacks. As reported in the OAuth 2.0 standard [21], the `state` parameter is defined as *"an opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client"*.

Fetch Metadata-Based. The Fetch Metadata request headers stand as a cutting-edge advancement in web platform security, designed to equip servers with potent defenses against cross-origin attacks. These headers, denoted as `Sec-Fetch-*`, provide vital contextual data about an HTTP request, granting the receiving server the ability to proactively enforce security measures before handling the request. This feature enables developers to make informed decisions about whether to approve or decline a request, taking into consideration its origin and intended scope. This methodology guarantees that only legitimate requests originating from their ~~own~~ application receive responses, thereby bolstering the overall security posture. There exist other header-based CSRF defenses including Referer and Origin header validation (interested readers can refer to [29], [5], [16], [34]).

SameSite Cookies-Based. `SameSite` is a cookie attribute defined in [14] that enables servers to specify whether a cookie should be included in cross-site requests. It offers three options: *Lax*, *Strict*, or *None*. When set to *Strict*, the cookie is withheld from all cross-site requests, even those initiated by regular links. On the other hand, the default *Lax* mode permits cookies for regular links from external sources but restricts them for CSRF-prone request methods like POST. *Lax* mode exclusively allows cross-site requests through top-level navigation and safe HTTP methods. Additionally, when set to *None*, it specifies that the cookie should be sent in all contexts.

User Interaction. As proposed in [3], this defense should recognize whether a request intentionally comes from the user or not. To do so, it requires involving the user in additional steps before accepting the request. These steps could include asking the user to perform authentication again, solving a captcha, or requesting consent at the IdP side for every SSO request. This type of defense can effectively prevent CSRF attacks but may have a high impact on the user experience.

2.3. Micro-Id-Gym

Micro-Id-Gym[1] (MIG) is an open source tool [8] used to assist system administrators and testers in the deployment and pentesting of Identity Management protocol deployment. In particular, the tool provides a plugin to support pentesting activities on SSO implementations. The pentesting tool is based-on Burp[2] web proxy and provides a good set of APIs to interact with. However, as we delve into fully automated testing, we encounter

1. https://st.fbk.eu/tools/Micro-Id-Gym.html
2. https://portswigger.net/burp

specific challenges, including dealing with obstacles such as captchas.

2.4. Obstacles to test automation

In web applications operating in production environments where captchas are enforced, automated UI testing can present significant challenges. This is particularly evident when captchas disrupt automated penetration testing activities. The concept of captchas inherently clashes with automation, as it is primarily designed to prevent automated bots from executing actions within the application. Similarly, multi-factor authentication (MFA) often requires user involvement, adding an extra layer of complexity to automated testing. These are common obstacles faced in automated testing. While these challenges may exist within production systems, it is less likely that they are present in the testing environments utilized by software testers at the SP. Otherwise, these systems would be not very testable. To work around these issues, we manually address them, assuming that testing systems on these websites do not feature the same automation hurdles.

3. SSO-based Linking Account

The SSO-based account linking process (SSOLinking, in short) allows users to link their accounts on a website (SP) to an SSO account they own at an IdP. As a consequence, users can log in on SP by leveraging popular IdPs for the authentication, while keeping their existing profiles on the SP. This ensures a SSO experience, besides the traditional form-based login.

Four entities take part in the protocol: the user U, controlling a web Browser, the SP and IdP. The main steps of the protocol are reported in Figure 1 and detailed below.

(1-3) At first, U logs in on the SP website, obtaining an authenticated session identifier on the SP website (say cookie(U,SP)). The big arrows in Figure 1 represent the multiple steps required for the login process.
(4) U opens the SSOLinking page on SP, selects the preferred IdP and clicks on the account linking button
(5) Browser sends an HTTP redirection request at SP with cookie(U,SP)
(6-7) SP redirects Browser to the IdP
(8) IdP provides the cookie policy and asks for the credentials
(9-10) U accepts the cookie policy and types the credentials
(11) IdP asks for the U's consent of sharing information with SP to link the U account at SP with the U account at IdP (e.g., the one reported in Figure 2)
(12-13) U provides the consent
(14-15) IdP redirects Browser to the SP making the user's browser send authentication data of the user on IdP (say code(U,IdP)) to the SP web site.
(16) SP confirms the linking of the U account at IdP, identified by code(U,IdP), with the U account at SP, identified by cookie(U,SP)

At the end of the process, SP links the user's IdP account with the existing user's SP account. This enables the user to login (via SSO) to the SP account using the IdP account.

Two variants of the protocol are possible. In case Browser has an authenticated session identifier on the IdP web site (say cookie(U,IdP)), in Step 7, cookie(U,IdP) is sent from Browser to IdP in the header of the request. As a consequence, steps 8-10 (marked with a red dashed rectangle in Figure 1) are not performed. Similarly, in case the consent has been already provided by the user in a previous execution, then steps 11-13 (marked with a blue dotted rectangle in Figure 1) are not performed.

Finally, what is presented here is the classical process, but there may be variants in the implementations on SPs. For instance, some SPs may use captcha and similar mechanisms to further protect the authentication phase. Moreover, as we will detail in Section 5.3, in some SPs, the account linking button does not trigger a redirection request to SP (i.e. step 5 of Figure 1), but a direct request to IdP, generated by using JavaScript from the IdP library at the SP side.

3.1. SSOLinking Account Hijack

In this paper, we focus on a severe account hijack (hereafter SSOLinking Account Hijack), reported in [22], that can be performed in the SSO-based account linking depicted in Figure 1.

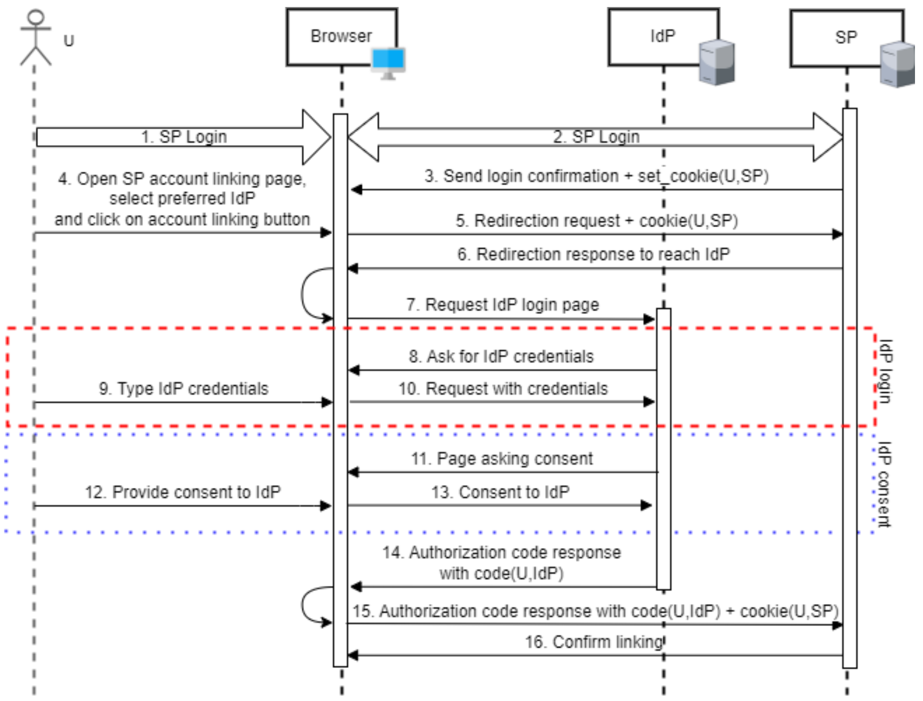
We consider a classical web attacker, which *(i)* owns/controls a (malicious) website (attack website), *(ii)* is capable to forge HTTP requests from the victim's web browser (e.g., by making the victim visit a link), and *(iii)* creates accounts at the target SP and the target IdP. In addition, we assume that the victim user has an active session on SP (obtained by authenticating themselves at the target SP through the traditional form-based login).

Then, we consider the following assumptions on SP and IdP:

(A1) IdP suffers from an Authentication CSRF vulnerability (a.k.a. login-CSRF, i.e., due to a missing CSRF protection, an attacker can force-login the victim to an attacker-controlled account on the IdP) and,
(A2) IdP-seamless: IdP has the SSO flow seamless (i.e., no IdP interface is shown) if the user has already logged in at the IdP. This means that steps 8-13 inside the red and blue rectangles in Figure 1 are not performed.
(A3) SP suffers from a SSOLinking-Init-CSRF vulnerability, namely the account linking button at the SP is vulnerable to CSRF.
(A4) SP enables U to login (via SSO) using the IdP account.

The steps to reproduce the attack are as follows. The attacker a plays the role of U and executes SSOLinking (by using its own credentials). In particular, it provides the consent to link an account at SP with its own account at IdP (step 12). Then the attacker unlinks his IdP account on his SP account.

The victim v—playing the role of U and having an active session with the SP (i.e. Browser owns cookie(v,SP))—visits the malicious website owned by the attacker and clicks on:


U
Browser
IdP
SP
1. SP Login
2. SP Login
3. Send login confirmation + set_cookie(U,SP)
4. Open SP account linking page, select preferred IdP and click on account linking button
5. Redirection request + cookie(U,SP)
6. Redirection response to reach IdP
7. Request IdP login page
8. Ask for IdP credentials
9. Type IdP credentials
10. Request with credentials
IdP login
11. Page asking consent
12. Provide consent to IdP
13. Consent to IdP
IdP consent
14. Authorization code response with code(U,IdP)
15. Authorization code response with code(U,IdP) + cookie(U,SP)
16. Confirm linking
Figure 1: The SSO-based account linking.

- a link (exploiting the Authentication CSRF vulnerability at IdP and IdP-seamless) to be (transparently) authenticated in the IdP as the attacker a (thus, setting cookie(a,IdP) on Browser of v);
- a link (exploiting SSOLinking-Init-CSRF vulnerability at SP) to send an HTTP redirection request to SP. As a consequence, Browser automatically includes cookie(v,SP) (step 5, Figure 1). Then, SP redirects Browser to IdP (step 7) and Browser sends cookie(a,IdP) to IdP (variant of steps 8-10)

As a result, the attacker a account at IdP, identified by code(a,IdP), is linked with the victim v account at SP, identified by cookie(v,SP).

In case (A4) holds, namely SP enables U to login via SSO using the IdP account (that is the attacker account), the impact of this attack is severe because the attacker's level of access to SP is identical to the legitimate user's. The attack results in a complete account hijack, meaning the attacker gains full control over the victim's account.

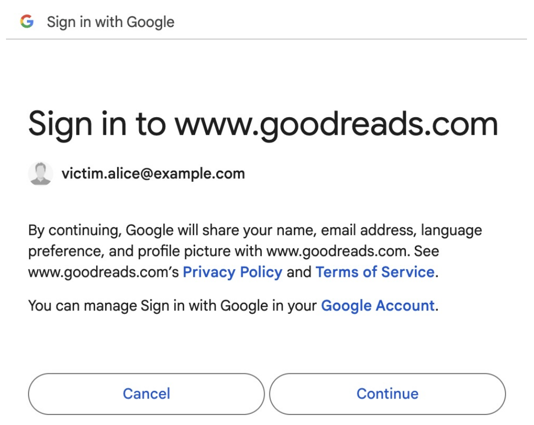
In addition, the attack remains "invisible": while preparing the links, the attacker should properly set the values of the `target` attribute (e.g., `_blank`), telling the browser to open the link on new (tiny) windows, which may easily go unnoticed by the user. Also, as revealed by our analysis, usually SPs neither notify users about other devices or active sessions nor allow users to see all the active sessions.

3.2. Comparison with another SSOLinking Account Hijack

It is interesting to note that there is a similar CSRF attack vector on the SSOLinking, leading an attacker to hijack the account of the victim. This attack is much more studied compared to the previous one (for instance described in Attack #10 in [34] and in [3]. The steps to reproduce this attack are as follows. The attacker a plays the role of U and executes SSOLinking (by providing its own credentials) until step 14 of Figure 1. Then, a intercepts the Authorization code response with code(a,IdP) and forces a victim user v (when the victim visits an attacker-controlled website) to send it to SP. Assuming that v has an active session with SP, Browser sends cookie(v,SP) to SP as well (cf. step 15). As a result, the account of a at IdP is linked with the account of the victim v at SP. This is exactly the same result of the attack reported in Section 3, but this attack exploits an incorrect CSRF protection in SSO callback endpoints (cf. Sections 2.2 and 9), while the previous one exploits a missing CSRF protection in the SSO initiation request (besides the Authentication CSRF affecting the IdP). The need of CSRF protection in SSO callback endpoints is well-known and SSO protocols already offer solutions for that (e.g., the `state` parameter in OAuth 2.0). On the contrary, missing CSRF protection in the initial request of SSOLinking, leading to SSOLinking Account Hijack, is understudied, as also pointed out by our study.

4. Our Approach

Exploiting the SSOLinking Account Hijack presented in the previous section relies thus on two vulnerabilities: an Authentication CSRF on the IdP side and an SSOLinking-Init-CSRF on the SSO-based account linking process on the SP side. The Authentication CSRF has been studied in various works [5], [34], [6] and many of such vulnerabilities have been reported, also to IdPs. However, in most of the cases the IdP opted to not fix such a vulnerability to allow users to make use of the one-click login feature. If Authentication CSRF vulnerabilities are not fixed, then investing further effort in developing testing techniques for this is not so worth. We validated



Sign in with Google

Sign in to www.goodreads.com

victim.alice@example.com

By continuing, Google will share your name, email address, language preference, and profile picture with www.goodreads.com. See www.goodreads.com's Privacy Policy and Terms of Service.

You can manage Sign in with Google in your Google Account.

Figure 2: Consent dialog shown by an IdP (Google) for an SP (Goodreads).

these two hypothesis—(I1) IdPs tend to be vulnerable to Authentication CSRF and (I2) IdPs tend to be reluctant to fix these vulnerabilities—in a specific experiment over the four major IdPs identified in [13]. The results of this experiment are detailed in Section 5.3.1 and confirm our hypothesis.

The CSRF vulnerabilities underlying our attack vector, when analysed in isolation within one party's (either SP's or IdP's) boundary, may be judged as less-critical. For instance, an IdP may consider Authentication CSRF vulnerabilities as low priority as the victim will likely notice that they are logged in at the wrong account. Similarly, the SP may consider SSOLinking-Init-CSRF as non-critical, as the IdP will show a consent dialog (similar to the one shown in Figure 2) and prevent the SSO flow from proceeding (for previously-unlinked IdP accounts). The underestimation of these vulnerabilities has led to an interesting and dangerous scenario where IdPs have been introducing features that aid Authentication CSRF and SPs have not been patching SSOLinking-Init-CSRF vulnerabilities. For instance, popular IdPs including Facebook, Instagram, and LinkedIn have introduced the one-click login feature that allows users to login to their IdP account by visiting a URL. These URLs are ideal payloads for Authentication CSRF attacks.

In our approach we thus assume that websites and, more specifically, IdPs can be vulnerable to Authentication CSRF. Under this assumption we design a testing technique to detect the SSOLinking Account Hijack on the SP side. Manual testing for this attack requires security expertise, is error-prone, and cannot be efficiently repeated on regular basis. In-line with CI/CD best-practices, we believe website developers conduct regular functional regression tests, anytime changes are implemented. Our vision is thus to empower developers with automated security tests upon the functional ones: developers create a functional test for SSOLinking, and SSOLinking Checker can then execute the security tests and accurately report vulnerabilities.

Figure 3 presents our approach. The Tester at the SP provides as input a Selenium script (s1) whose user actions (e.g., click on a button) are run to execute SSOLinking. The HTTP messages (requests and responses) generated while executing the user actions are collected by the Proxy and retrieved by our checker via the Proxy API. Next, the "Attack steps inference" module infers, from the execution of (s1) and from the retrieved HTTP messages, the attack steps to be run to detect the SSOLinking Account Hijack at this specific SP. These attack steps are saved as a new Selenium script (s2). Among the actions in (s2) some relate with the IdP used for SSOLinking. These are created by the "Attack steps inference" module by means of the "IdPs metadata", a little database populated offline by us with the metadata of the major IdPs and easily extendible for other IdPs. Also, one of the action in (s2) requires an HTML form to send a specific cross-site HTTP request to the SP. This form is generated by the "HTML POC Generator" module. If the attack steps in (s2) are all successfully executed, then the SP is reported vulnerable to the Tester. Besides presenting the verdict for the attack (successful or not), the output also comprises a proof-of-concept of the attack (if the attack was successful), and a log file with the entire HTTP traffic. This information enables the Tester to drill-down into the details of the test as well as to reproduce the attack.

We provide hereafter more details about the key parts of our approach and its implementation within our SSOLinking Checker.

4.1. Input Selenium Script

The input Selenium script (s1) is just a UI integration test that testers are used to create for web applications [32]. Indeed, (s1) just comprises the user actions to functionally execute with the Browser the entire SSOLinking process in an automated way at anytime, so to be certain the process is working as expected. We can decompose (s1) in 4 main parts: Login at SP, Link accounts, Assert, and Unlink accounts. Let us discuss each one of them, also presenting some real examples.

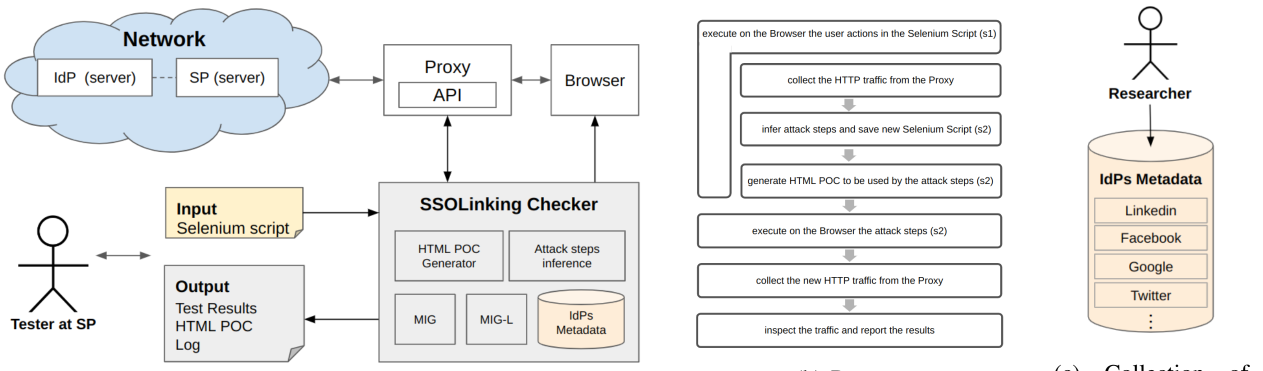
Login at SP. This comprises the user actions to login a testing user at the SP (cf. step 1 in Figure 1). We illustrate an example for the Daily Mail website in Listings 1. The Daily Mail login page is opened and the cookie policy accepted. Then email and password are filled-in and the login button clicked.

```
open | https://www.dailymail.co.uk/registration/login.html |//opens login page
click | xpath=/html/.../div/button[2] | //accepts the cookie policy
click | xpath=/html/.../div[2]/input | //clicks on email text field
type | xpath=/html/.../div[2]/input | john@example.com //types the email
click | xpath=/html/.../div[3]/input | //clicks on password text field
type | xpath=/html/.../div[3]/input | 12345678 //types the password
click | xpath=/html/.../div[5]/button | //clicks the login button
```

Listing 1: Login at SP - Daily Mail example.

Link accounts. This includes the user actions to link the testing user account as authenticated at the SP with the one at the IdP (cf. steps 4, 9, and 12 in Figure 1). We continue the example for the Daily Mail website (SP) when linking accounts with Twitter (IdP) in Listings 2. The Selenium commands are identical to the ones in the previous listing and the comments describe each user action. Note, however, that the credentials of a testing user at the IdP are provided within the actions and easily retrievable in our approach. We use `<email-idp>` and `<password-idp>` to refer to these retrieved credentials.

```
open | https://www.dailymail.co.uk/registration/profile/edit.html | //opens
      account page
click | xpath=/html/.../li[1]/a | //clicks on Twitter link account button
```


Network
IdP (server)
SP (server)
Proxy
API
Browser
Input
Selenium script
SSOLinking Checker
HTML POC Generator
Attack steps inference
MIG
MIG-L
IdPs Metadata
Tester at SP
Output
Test Results
HTML POC
Log
execute on the Browser the user actions in the Selenium Script (s1)
collect the HTTP traffic from the Proxy
infer attack steps and save new Selenium Script (s2)
generate HTML POC to be used by the attack steps (s2)
execute on the Browser the attack steps (s2)
collect the new HTTP traffic from the Proxy
inspect the traffic and report the results
Researcher
IdPs Metadata
Linkedin
Facebook
Google
Twitter
⋮

(a) Architecture (b) Process (c) Collection of IdPs metadata.

Figure 3: High level view of our approach.

```
click | xpath=/html/.../button[2] | //accepts Twitter cookies
click | id=email | //clicks on email text field
type | id=email | john@abcd.com //types the email of user on IdP, <email-idp>
click | id=pass | //clicks on password text field
type | id=pass | abcdefgh //types the password on IdP, <password-idp>
click | id=loginbutton | //clicks the login button
click | xpath=/html/.../div | //clicks to give consent to link accounts
```

Listing 2: Link accounts - Daily Mail with Twitter.

Assert. Now that the linking between accounts should be done, the Tester wants to be sure this was completed successfully. Assertions provide an easy way to set the expectations for the UI test. Listing 3 presents the assertions for our example on linking accounts between Daily Mail (SP) and Twitter (IdP). The idea is very simple: the account profile is accessed and the fact that the linking button with Twitter is not there allows to derive that the linking was already done.

```
open | https://www.dailymail.co.uk/registration/profile/edit.html | //re-opens
        account page for @assertion purposes
assert not clickable | xpath=/html/.../div/a | //verifies the not availability
        of the account linking button with Twitter
```

Listing 3: Assert - Daily Mail with Twitter.

Unlink accounts. The Input Selenium Script shall be repeatable, so to enable the Tester to test the process anytime. To do so (s1) is completed with a reset process to unlink the accounts. Indeed, if the accounts stay linked, then the previous parts of the script would simply fail. Listing 4 completes our script example and illustrates the user actions to unlink the user account at Daily Mail from the account on Twitter.

```
open | https://www.dailymail.co.uk/registration/profile/edit.html | //re-opens
        account page
click | xpath=/html/.../div/a | //clicks unlink account button for Twitter
click | xpath=/html/.../span/a[2] | //confirm to unlink the accounts
```

Listing 4: Unlink accounts - Daily Mail with Twitter.

4.2. IdPs metadata

Our approach relies on a few IdP-related metadata. This is an offline activity that requires little effort and can be shared among few researchers to collect metadata for many IdPs (cf. Figure 3-(c)).

```
host: twitter.com
redirect: [/o/oauth2/auth]
login:
open | https://www.twitter.com | //access Twitter
click | xpath=/html/body/div[3]/.../button[2] | //clicks to authenticate
click | id=email | //clicks on email text field
type | id=email | <email-idp> //types the email
click | id=pass | //clicks on password text field
type | id=pass | <password-idp> //types the password
click | id=loginbutton | //clicks the button
```

Listing 5: IdP metadata - Twitter.

The procedure to support new IdPs is straightforward and amount to append in the IdPs metadata database three IdP information. These information are described hereafter and a complete example is presented in Listing 5 for the Twitter IdP.

(1) IdP Host (`host`). Simply the host of the IdP.

(2) IdP Redirection Signature (`redirect`). This is the list of pattern signatures that our approach uses to detect the redirection response that the SP generates to reach a specific IdP host. In most of the cases, this metadata entry will be something like `dialog/oauth`. It is referred as macro `<IdP_redirection>` in lines 3 and 5 of Listing 6.

(3) IdP Login (`login`). This comprises the Selenium actions to login at the IdP. It is referred as macro `<IdP_login_act>` in Listing 6. The user credentials are not hardcoded, but they are rather specified as macros. We recall that our approach automatically extracts them from the "Link Account" fragment of (s1).

4.3. Infer the attack steps

While the Selenium script (s1) is executed, the "Attack steps inference" module infers the attack steps to be run to detect the SSOLinking Account Hijack at a specific SP and saves them in a new Selenium script (s2). The inference relies on a new component MIG-L that we introduced to extend the open-source platform MIG [7]. MIG-L defines a specification language we extended to execute tests in MIG where, e.g., new Selenium scripts can be created and run on-the-fly by processing the execution of other Selenium scripts.

```
1  run($s1) //runs Selenium script (s1)
2  // Inference begin
3  mark($L,<IdP_redirection>)
4  save_act([1,index($L)),$SP_login_act)
5  save_msg(<IdP_redirection>,$SP_acc)
6  save_host($SP_acc, $IdP_host)
7  save_poc($SP_acc,$POC)
8  mark($A,contains{assert, @assertion})
9  save_act($A,$SP_assert)
10 add($s2,$SP_login_act)
11 add($s2,get_idp_login($IdP_host, <IdP_login_act>))
12 add($s2,"open | http://localhost/"+$POC)
13 add($s2,"click | id=SSOLinking")
14 add($s2,"wait | 2000")
15 add($s2,$SP_assert)
16 // Inference end
17 run($s2) //runs Selenium script (s2)
```

```
18  result($s2) //compute the results: vulnerable if (s2) successfully executed
```
Listing 6: Test procedure for SSOLinking Account Hijack with inference procedure.

For instance, Listing 6 is the procedure written in MIG-L to test the entire SSOLinking Account Hijack at any SP (commands are presented in red, macros in purple, and comments in blue).

The first `run` command just indicates that the Selenium script (s1) is executed. Then, the inference procedure starts and proceeds as following line by line:
Line 3. The inference procedure observes the HTTP messages and marks with `L` the user action of (s1) whose corresponding HTTP messages comprise the value of the macro `<IdP_redirection>`. The macro is extracted from the IdPs metadata database.
Line 4. All the user actions in (s1) from the first till the one marked with `L` are saved into the variable `$SP_login_act`. Note that closing round bracket indicates that the action marked with `L` is excluded. All these actions capture the SP login activity.
Lines 5-7. The first HTTP message request whose HTTP response comprises the macro `<IdP_redirection>` is saved into variable `$SP_acc`. The host to which the response is redirected is saved into `$IdP_host` and then an HTML POC is generated by the "HTML POC Generator" module. This HTML POC, whose filepath is saved into variable `$POC`, enables later on the sending of a cross-site request mimicking the original one (more details in Section 4.4).
Lines 8-9. All user actions in (s1) that contain any of the keywords `assert` or `@assertion` as a comment are marked with `A` and then saved in variable `$SP_assert`.
Lines 10-15. The new Selenium script (s2) can now be built. First, all the user actions related to SP login (saved into `$SP_login_act`) are added. Then, the user actions related to IdP login are added. These are simply extracted from the IdP metadata by selecting the right IdP host `$IdP_host`. We refer to these actions as `<IdP_login_act>`. Then three Selenium commands are also appended into (s2): the first will open the HTML POC local page (notice that the path depends on `$POC`), the second will run the cross-site request, and the third will just add some waiting time to ensure the request is properly processed. Last, but not least, the assertions collected in `$SP_assert` are added to (s2).

Now that the inference is completed and (s2) is created, the test procedure simply proceeds in running (s2) in line 17 and checking its result in line 18. If all the Selenium actions in (s2) are successfully executed, then the SP is vulnerable (more details in Section 4.5).

4.4. Generate the HTML POC

This module generates a proof-of-concept form able to craft the proper CSRF exploit to probe the SSOLinking-Init-CSRF. In details, it creates a HTML page with a form to send the HTTP request in `$SP_acc` properly changed. Indeed, according to the HTTP method of `$SP_acc`, the module builds the page and the HTML form. In case of a GET message, the form makes a simple request to the specific URL, while for a POST message the content-type and body are added.

4.5. Output

The output of our testing approach comprises three elements. The first element is the test result that provides the verdict whether the attack was successful or not. If all the user actions in (s1) and those created for (s2) are successfully executed, then the attack is successful. If some user actions of (s2) were unsuccessful (e.g., an unexpected page is displayed, an element which should be clicked is not present in the page), then the attack is unsuccessful. The second element of the output is the HTML POC useful to reproduce the attack (when the attack was reported). The last element is a log file with all the HTTP messages saved during the execution of the test. The key to avoiding false positives and false negatives is by correctly defining the Selenium assertions that determines successful SSOLinking (see Section 4.1). Having said that, some tests may still fail because of network problems, CAPTCHA challenges, and other reasons, but such occurrences are detectable within our approach.

4.6. Implementation

We implemented our approach in SSOLinking Checker. It is as an extension of Micro-Id-Gym [7] programmed in Java and uses the API of the widely-used penetration testing tool Burp to perform standard proxy engine operations such as collecting HTTP traffic to search for specific HTTP message, setting proxy rules to alter the HTTP traffic, etc. To specify the procedure to infer the attack steps, we used a JSON version of MIG-L, which is an equivalent variant currently supported by MIG of the one described in Section 4.3. The source code, installation guide and tutorial of our prototype are publicly available in [1].

5. Experimental analysis

We demonstrated the effectiveness of our approach and the pervasiveness of the SSOLinking Account Hijack against a selection of popular websites featuring the SSO-based account linking process with major IdPs (48 unique pairs SP-IdP). In doing so, our approach discovered 21 vulnerable unique pairs SP-IdP out of the 48 analyzed. In the rest of this section, we detail the selection of the major IdPs and popular websites (see Section 5.1) that we used in our experiments, the methodology used for our experiments (see Section 5.2), and the results obtained (see Section 5.3).

5.1. Dataset selection

We selected 4 major IdPs and for each of them 12 SPs supporting the SSO-based account linking process and login via SSO using the IdP account. Our selection leverages the dataset released by [13] in 2018, where the authors crawled the most popular one million websites to infer whether they integrate the SSO login process. This is very valuable information for our study as websites featuring the SSO-based account linking process are clearly a subset of those having SSO login. Even if some of the data from [13] is clearly outdated, this did not impact our study and we were able to select our candidates easily.

Listing 7 reports an example of an entry of the dataset from [13]. Each entry specifies in the field `sso` which IdPs that website is using (if any). For this example, `workable.com` is inferred to have SSO login with Google and LinkedIn IdPs.

```
{ "rank": "4824"
  "url": ["https://www.workable.com/signin",
          "http://workable.com",
          "https://www.workable.com/signup",
          "https://www.workable.com/"],
  "sso": ["google", "linkedin"],
  "redir_from": [] }
```

Listing 7: Entry (excerpt) from [13].

For each IdP mentioned in the dataset we counted how many websites were making usage of them in the 2018 dataset and selected the 4 IdPs with the highest number of occurrences. Table 1 presents the IdPs candidates and the number of occurrences reported for each of them. The selected IdPs for our experimental analysis are then Facebook, Google, Twitter and Linkedin.

For each one of the selected IdPs, we then proceeded in selecting 12 SPs having SSO-based account linking with the IdP. We considered one by one the website entries in their ranking order from the 2018 dataset for which one of our selected IdP occurred. We manually checked whether the SSO-based account linking process was implemented by that website with the IdP. As shown in Table 2, we manually analyzed a corpus of 648 SPs to discover 67 websites supporting the SSO-based account linking process with the selected IdPs. Among the 67 SPs, we discarded 19 websites due to web issues that occurred in the execution of the process (14) and automation detection issues (5). With web issues, we refer to issues in the SP website that leads to e.g., broken pages and links during the account linking or unlinking processes. For automation detection issues we mean those SPs that e.g., identify the use of Selenium libraries to automate browser actions as a bot issue, which denies access to the website. The SPs selection procedure is successfully finished by obtaining 12 fully working SPs for each selected IdP. Some of these SPs are used in our dataset for more than one IdP. For instance, sp4 occurs in our dataset as paired with Facebook, Google, and Twitter IdPs. In total, we tested 48 SSOLinking processes (i.e. unique pairs SP-IdP) including 37 different SP websites.

5.2. Methodology

Here we describe the methodology we followed to perform the experiments on our selected dataset.

TABLE 1: IdP candidates.

IdP Name	Occurrences
Facebook	43,333
Google	26,186
Twitter	12,465
LinkedIn	3,939
Amazon	1,459
Yahoo	1,924

TABLE 2: SPs selection per IdP.

	SPs per IdP	**SPs with SSOLinking**	**SPs with issues**	
			Web	Autom.
Facebook	61	18	4	2
Google	128	12	0	0
LinkedIn	371	24	9	3
Twitter	88	13	1	0
Total	**648**	**67**	**14**	**5**

First of all, we focused our attention to validate the two hypothesis (I1) and (I2) we made in our approach for IdPs (cf. Section 4). We manually analysed the four IdPs in our dataset against the Authentication CSRF and discovered that three of them are vulnerable and that they do not plan to fix the problem. More details are given in Section 5.3.1.

While analysing the IdPs we also collected the few IdPs metadata required for our approach (cf. Listing 5) which provides a concrete example of the IdP metadata for Facebook. We make available the other IdPs metadata for Google, Twitter and LinkedIn in [1].

The last step in our methodology aims to evaluate the pervasiveness of SSOLinking Account Hijack. In this respect, we run our SSOLinking Checker against the 48 pairs of SPs and IdPs in our dataset. For each pair of SP and IdP, we played the role of a tester at the SP and we created the Selenium script to execute the SSO-based account linking process.

In doing so we manually performed the registration of users on the SP websites and we followed the steps described in our approach (see Section 4). For the purpose of the experimental analysis, namely to simplify the manual effort being able to increase the number of analysed SPs, we considered the challenging option to leverage state-of-the-art automation tools to automatically perform the registration of the users and the execution of the SSOLinking for generating the selenium script. Initially, we explored Shepherd [15], a tool for basic website authentication. However, it lacked support for automated registration and relied on credentials available or leaked on the web, rendering it unsuitable for our experiments. Ultimately, we experimented with the xdriver-open [11], a tool also designed to assist in website authentication but the tool was not successful in completing registrations on most websites in our dataset [13].

All the scripts are available in [1]. Our SSOLinking Checker detected the account hijack in almost 50% of our dataset. More details are given in Section 5.3.2, while responsible disclosures and confirmations from vulnerable SPs are discussed in Section 7. All in all, our results indicate that this attack may really be overlooked by the web community.

5.3. Results

We report here the results obtained by following the experiments described in our methodology.

5.3.1. Manual testing of the IdPs. We analysed manually the four IdPs in our dataset to validate our two hypothesis: (I1) IdPs tend to be vulnerable to Authentication CSRF;

and (I2) IdPs tend to be reluctant to fix these vulnerabilities.

We detected the vulnerability in 3 over 4 of our IdPs, namely Facebook, Google and LinkedIn. The details are below for each vulnerable IdP. We reported the vulnerability to the vendors, but even if they recognize it they do not plan to fix the issue. We responsibly disclosed Authentication CSRF to IdPs, along with concerns regarding SSOLinking Account Hijack on SP side. Unfortunately, these concerns were dismissed by the IdPs and categorized as intended features to reduce login burden. The risk is partially mitigated with login-tokens whose validity ranges from 5 min (Google) to 15 min (LinkedIn). However, an attacker can overcome this by programmatically requesting a fresh login token and injecting the attack payload after detecting the victim's visit to the attacker-controlled website. Additionally, Google implemented a defense within the Chrome browser (see Section 6). However, this defense proved ineffective in preventing the Authentication CSRF attack on Google using Chrome in our experiments.

Facebook. The following steps enable the exploitation of Authentication CSRF in Facebook (let `<FB>` be the url `https://www.facebook.com/recover`):

1) the attacker opens the link `<FB>/initiate/` in a browser;
2) the attacker provides his facebook account email;
3) the attacker enters the code received by email;
4) the attacker presses the "continue" button;
5) the attacker intercepts and drops the HTTP GET request to this url `<FB>/password/?u={UCODE}&n={CODE}&fl=default_recover&sih=0&msgr=0` saving the parameter's value `<UCODE>` and `<CODE>` respectively;
6) the attacker forges the following URL `<FB>/password/?<QS>` where the query string `<QS>` comprises `u=<UCODE>`, `n=<CODE>`, `ars=one_click_login`, `fl=one_click_login`, `spc=1`, `ocl=1`, `sih=0`, and `msgr=0`;
7) the attacker lures the victim to click on the link created at the previous step;
8) the victim is now authenticated in Facebook as the attacker.

Google. Calzavara *et al.* [9] managed to mount an attack against Google that allows a web attacker to authenticate any user on Google under the attacker's account. Once accessed, a malicious web page can cause a victim's browser to issue the attacker's request to the Google assertion consumer service (based on SAML 2.0), thus forcing the victim inside the attacker's controlled authenticated session. The vulnerability can be exploited by any attacker with a valid account on a third party IdP that uses Google as SP. This is a very common business scenario that allows the users of that IdP to consume services such as GMail and Google Calendar. We leveraged this vulnerability to gain access to a Google account. Indeed, once authenticated to the Google SP as the attacker, the victim is now authenticated as the attacker in the entire Google realm, even when Google serves as IdP other SPs. We tested this scenario and the vulnerability is still exploitable.

LinkedIn. To exploit the Authentication CSRF in LinkedIn the following steps are sufficient:

1) the attacker opens in a browser the following link `https://www.linkedin.com`;
2) the attacker performs login using his credentials;
3) the attacker clicks "Sign Out" button on the profile page;
4) the attacker opens the URL `https://www.linkedin.com/login?trk=homepage-basic_intl-segments-login` and clicks the button "Sign in with one-time link";
5) an email with a link is sent from LinkedIn to the attacker email account to login with one click;
6) the attacker lures the victim to click on that link;
7) the victim is now authenticated in LinkedIn as the attacker.

Twitter. In our analysis, we also considered Twitter as potentially vulnerable to Authentication CSRF, given its tendency to exhibit vulnerabilities according to previous studies [34], [4].

5.3.2. Testing of the SPs.

Here we report about the experiments performed on detecting SSOLinking-Init-CSRF on the SPs in our dataset. For the vulnerable SPs who have not informed us of having patched the vulnerability, we have employed aliases (e.g., sp1, sp2) instead of using the actual SPs names. By experimenting with our SSOLinking Checker on our dataset, which comprises 48 unique pairs SP-IdP implementations supporting the SSO-based account linking process, we found that 21 out of the 48 analyzed (43.7%) were reported as vulnerable. Interesting enough, we encountered only two instances of the test automation problems discussed in Section 2.4. For two SPs, namely Naver and sp4, our SSOLinking Checker required manual intervention to overcome a captcha and a two-factor authentication check.[3] Table 3 summarizes our results per IdP, indicating how many SPs in [13] are vulnerable, along with their popularity ranges in the Tranco list [28] updated as of October 2023.

Overall, the vulnerable SPs distribute uniformly over the different IdPs, indicating that there is not an IdP providing "better" SDK and documentation to ensure a secure integration procedure for SPs.

This is even clearer from the detailed results presented in Table 4. For each pair of SP and IdP, we first show

3. As discussed in Section 2.4 these are classical obstacles to testing automation. While these obstacles may be present in the productive systems (the one we could analyse), they are likely not used in the testing systems that a tester at the SP would deal with. We solved those obstacles manually, assuming that in the testing systems of those websites the testing automation obstacles would not be there.

TABLE 3: Overview of the SP test results per IdP.

	SPs per IdP		**Tranco range**
	Vulnerable	Not Vulnerable	First-Last
Facebook	5	7	84-1506
Google	4	8	43-17482
LinkedIn	6	6	3834-314117
Twitter	6	6	84-18293
Total	**21**	**27**	**-**

some insights of their implementations, namely (i) the IdP library, the endpoint at the IdP side invoked by the SP for the SSOLinking process (see column **IdP Library**) and (ii) whether the SP employs a redirection request (cf. step 5 in Figure 1) for the SSOLinking process or not (see column **Redirection Request**). Column **Vuln.** indicates the result of our experiment indicating whether the SP is vulnerable to SSOLinking-Init-CSRF or not. For SPs identified as not vulnerable, we provide information on the countermeasures implemented (see columns **Protection**), such as User Interaction (U), Token-Based (T), Fetch Metadata-Based (M), Referer (R), Origin (O) and SameSite-Based (S). (The most recent protections are discussed in Section 2). To test whether an SP implements U defense to prevent the exploitation SSOLinking-Init-CSRF, we simulated the action that initiates the SSOLinking process at the SP and observed its responses. Specifically, we analyzed whether the SP requires the user to perform additional actions, such as re-authentication, resolving a captcha, confirming association, or providing consent before completing SSOLinking process. For testing the implementation of the T defense, we first obtained an outdated CSRF token from another user and inserted into a new session that attempts to exploit SSOLinking-Init-CSRF. If successful, this outcome indicates that the CSRF token implementation may be inadequate. To determine whether M defense is implemented, the values of the Fetch Metadata headers in the HTTP request, which correspond to initiating the SSOLinking process at the SP, were replaced with values simulating a cross-site context (e.g., `Sec-Fetch-Site: cross-site` and `Sec-Fetch-Mode: navigate`). If the HTTP request fails, it indicates that the SP is implementing M defense. Regarding verifying the implementation of the R and O defenses, the Referer (or Origin resp.) header's value was intentionally altered in the HTTP request initiating the SSOLinking process at the SP to emulate a cross-site context. If the manipulation does not block the execution of the SSOLinking process, it indicates a lack of implementation of the defense at the SP. Finally, to evaluate the implementation of the S defense, we verify whether the SP is indeed adding the `SameSite` attribute to cookies, such as `Lax` and `Strict`, in the SSOLinking request. Note that we also analysed the outcomes of our SSOLinking Checker manually, as a tester at the SP would do to be certain of the verdicts. This allowed us to ensure the correctness of our approach in the dataset. We checked the IdP library used by each SP to determine if the vulnerability is somehow related to the IdP library used by the SP. From the results, there is no clear correlation between the adopted IdP library and the vulnerability at the SP. This particular analysis was conducted to determine whether a particular version of IdP or SDK libraries is vulnerable by design. More interesting, the fact that the few SPs (e.g., Mediafire) not adopting the redirection request are all not vulnerable. These SPs uses an account linking button triggering a direct request to the IdP (i.e., no redirection request), generated by using the JavaScript of the IdP Library. Also interesting the fact that the same website features both vulnerable and not vulnerable SP implementations. For instance, sp2, a popular news website, implements a vulnerable SP in linking accounts with Facebook while the one with Twitter is not vulnerable. Similarly, sp1, a popular image-sharing website, is vulnerable with Facebook and not with Google. sp6, a famous Southeast Asian forum website, is systematically vulnerable with both Google and Twitter. The same for sp3, a widely-used blogging website, with both Facebook and Twitter, and sp4, a renowned video-sharing website, with all Facebook, Google, and Twitter. Stackoverflow and Yandex emerges as those well protected with all their IdPs. As for the protection, most of the non-vulnerable SPs use a Token-Based defense with some notable exceptions for Qiita that also employ Origin, SameSite and Fetch Metadata-Based ones. All in all, we do not observe a clear explanation separating the websites that are vulnerable from those that are not. However, the further analysis that we performed on the `state` parameter may shed some lights. The results of this analysis are presented in the last column of Table 4 (see column **`state` vuln.**). For each pair of SP and IdP we inspected and tested the `state` parameter within the SSO-based account linking process. This `state` parameter is key for this kind of SSO processes and its misuses—using an empty value or removing the parameter altogether—have been discussed and advertised a lot in the literature and in the community. Those misuses, if not properly considered and protected, could lead to the very same CSRF attack that we have been tested with our approach using a different attack vector. Very interesting, most of the SP implementations—namely 19 over 21—that we found vulnerable with our approach are well protected against misuses of the `state` parameter. In our opinion, this last test clearly indicates that when the web community is alerted multiple times about the risks of certain vulnerabilities, counter-measures are taken in most of the cases. This also indicates that the attack vector we have been targeted and analysed may really have been overlooked by the web community so far and that more advertisement is required to get the right protection measures in place. We successfully exploited the SSOLinking Account Hijack attack in all 21 vulnerable SP-IdP pairs to confirm the accuracy of our approach. Exploit verifications were conducted using different networks, devices, and browsers for both the victim and the attacker.

6. Mitigations

We distinguish two categories of mitigations for SSOLinking Account Hijack. The former category includes mitigations that can prevent SSOLinking Account Hijack, while the latter category includes mitigations that allow the victim to detect the attack, once it has already been successfully executed. As already mentioned, SSOLinking Account Hijack allows the attacker to gain full control over the victim's SP account.

6.1. Mitigations for prevention

In Section 3.1, we listed the assumptions (A1), (A2), (A3), and (A4) paving the way to SSOLinking Account Hijack. To prevent the attack, it is therefore sufficient that at least one of these three assumptions is not fulfilled.

Concerning (A1), namely IdP suffers from an Authentication CSRF vulnerability, the mitigation consists in implementing a CSRF protection at the IdP (e.g., one of

TABLE 4: Testing of the SPs: complete results.

SP	**IdP**	**IdP Library**	**Redirection Request**	**Vuln.**	**Protection**						**`state` vuln.**
					U	**T**	**M**	**R**	**O**	**S**	
sp1	Facebook	/v8.0/dialog/oauth	yes	yes	-	-	-	-	-	-	no
sp2	Facebook	/v8.0/dialog/oauth	yes	yes	-	-	-	-	-	-	yes
fandom.com	Facebook	/dialog/oauth	yes	no	-	-	✓	-	-	✓	no
mediafire.com	Facebook	/v2.9/dialog/oauth	no	no	N/A	N/A	N/A	N/A	N/A	N/A	no
sp3	Facebook	/v5.0/dialog/oauth	yes	yes	-	-	-	-	-	-	no
naver.com	Facebook	/v2.12/dialog/oauth	yes	yes	-	-	-	-	-	-	no
sp4	Facebook	/dialog/oauth	yes	yes	-	-	-	-	-	-	no
quora.com	Facebook	/v13.0/dialog/oauth	no	no	N/A	N/A	N/A	N/A	N/A	N/A	no
stackoverflow.com	Facebook	/v2.0/dialog/oauth	yes	no	-	✓	-	-	-	✓	no
yandex.ru	Facebook	/v10.0/dialog/oauth	yes	no	✓	-	-	-	-	-	no
airbnb.com	Facebook	/v10.0/dialog/oauth	yes	no	-	✓	-	-	-	✓	no
asos.com	Facebook	/v10.0/dialog/oauth	yes	no	✓	-	-	-	-	✓	no
9gag.com	Google	/o/oauth2/auth	yes	no	-	-	✓	-	-	✓	no
sp5	Google	/o/oauth2/v2/auth	yes	yes	-	-	-	-	-	-	no
bestbuy.com	Google	/o/oauth2/auth	no	no	N/A	N/A	N/A	N/A	N/A	N/A	no
goodreads.com	Google	/o/oauth2/auth	yes	yes	-	-	-	-	-	-	no
sp6	Google	o/oauth2/auth	yes	yes	-	-	-	-	-	-	yes
sp4	Google	/o/oauth2/auth	yes	yes	-	-	-	-	-	-	no
pinterest.com	Google	/v3/signing/	no	no	N/A	N/A	N/A	N/A	N/A	N/A	no
stackoverflow.com	Google	/o/oauth2/auth	yes	no	-	✓	-	-	-	✓	no
yandex.ru	Google	/o/oauth2/auth	yes	no	✓	-	-	-	-	-	no
reverso.net	Google	/o/oauth2/auth	yes	no	-	✓	-	-	-	✓	no
dmm.com	Google	/o/oauth2/auth	yes	no	-	✓	-	-	-	✓	no
wix.com	Google	/o/oauth2/auth	yes	no	✓	-	-	✓	-	✓	no
allaboutcircuits.com	LinkedIn	oauth/v2/authorization	yes	no	✓	-	-	✓	-	✓	no
sp7	LinkedIn	oauth/v2/authorization	yes	yes	-	-	-	-	-	-	no
sp8	LinkedIn	oauth/v2/authorization	yes	yes	-	-	-	-	-	-	no
clubedohardware.com.br	LinkedIn	oauth/v2/authorization	yes	no	-	✓	-	-	-	✓	no
codementor.io	LinkedIn	oauth/v2/authorization	yes	no	-	-	✓	-	-	✓	no
docsend.com	LinkedIn	oauth/v2/authorization	yes	no	-	✓	-	-	✓	✓	no
sp9	LinkedIn	oauth/v2/authorization	yes	yes	-	-	-	-	-	-	no
workable.com	LinkedIn	oauth/v2/authorization	yes	yes	-	-	-	-	-	-	no
cpjobs.com	LinkedIn	oauth/v2/authorization	no	no	N/A	N/A	N/A	N/A	N/A	N/A	no
jobs.oxfam.org.uk	LinkedIn	oauth/v2/authorization	yes	no	-	-	-	✓	-	✓	no
sp10	LinkedIn	oauth/v2/authorization	yes	yes	-	-	-	-	-	-	no
sp11	LinkedIn	oauth/v2/authorization	yes	yes	-	-	-	-	-	-	no
dailymail.co.uk	Twitter	/o/oauth2/auth	yes	no	✓	-	-	-	-	✓	no
sp6	Twitter	/oauth/authenticate	yes	yes	-	-	-	-	-	-	no
mediafire.com	Twitter	/oauth/authenticate	yes	no	-	-	✓	-	-	✓	no
sp3	Twitter	/oauth/authenticate	yes	yes	-	-	-	-	-	-	no
sp12	Twitter	/oauth/authenticate	yes	yes	-	-	-	-	-	-	no
sp4	Twitter	/oauth/authenticate	yes	yes	-	-	-	-	-	-	no
sp13	Twitter	/oauth/authenticate	yes	yes	-	-	-	-	-	-	no
qiita.com	Twitter	/oauth/authenticate	yes	no	-	✓	✓	-	✓	✓	no
yandex.ru	Twitter	/oauth/authenticate	yes	no	✓	-	-	-	-	-	no
habrahabr.ru	Twitter	/oauth/authenticate	yes	no	-	-	-	✓	-	✓	no
sp14	Twitter	/oauth/authenticate	yes	yes	-	-	-	-	-	-	no
pixiv.net	Twitter	/oauth/authenticate	no	no	N/A	N/A	N/A	N/A	N/A	N/A	no

Legend: "U": User Interaction, "T": Token-Based, "M": Fetch Metadata-Based, "R": Referer-Based, "O": Origin-Based, "S": SameSite-Based

the defenses for CSRF outlined in Section 2.2, apart from "User interaction" which will be considered in assumption (A2)). This may disable the one-click login feature offered by IdPs and weaken the user experience.

Concerning (A2), namely IdP has the SSO flow seamless, the mitigation consists in implementing the "User interaction" defense. For instance the IdP displays a consent dialog to users. One option is to show the consent dialog during any authentication/authorization process. However, this would significantly impact the user experience. For OAuth, we suggest the SP use the `prompt` parameter set to `consent` in the authorization request in case of SSOLinking. This setting is supported by the IdP's libraries/SDK, and it ensures that users are explicitly prompted for permission during the authorization process. Since the SSOLinking process does not occur frequently, the impact of this mitigation on the user experience is limited.

As an alternative, the consent dialog can be triggered by the browser itself. As part of this mitigation, users will be redirected to a new confirmation page on the IdP to validate their identity. To minimize user disruption and improve the user experience, the confirmation page will only be presented once per account per device. Nevertheless, this mitigation is specifically designed for users logging in to Google Suite services using the Chrome web browser with SAML authentication[4].

Concerning (A3), namely SP suffers from a SSOLinking-Init-CSRF vulnerability, similarly to the previous cases, the mitigation consists in implementing a CSRF protection at the SP (e.g., one of the defenses for CSRF outlined in Section 2.2). However, differently from the previous cases, the user experience would not be affected by this fix.

This is the reason why in this work we provide our automated testing technique: it assists a tester at the SP to easily detect SSOLinking-Init-CSRF, and then the issue can be fixed by implementing one of the common CSRF defenses.

6.2. Mitigations for detection

The aim of mitigations for detection is to make users aware of the relevant actions performed on their accounts, allowing them to detect the ones executed unintentionally.

The two relevant actions performed to execute the SSOLinking Account Hijack are first the (forced) login of the victim on the attacker controlled account on the IdP, and second the successful execution of the SSOLinking.

For both the actions, sending notifications to the IdP account upon successful login and SSOLinking would not mitigate the issue as the attacker has full control of the IdP account. On the contrary, the SP could notify the victim's account of a successful SSOLinking (e.g., by sending an email from the SP to the victim's email retrieved from their account in the SP). This mitigation might be helpful for the victim to detect the SSOLinking Account Hijack. However, the burden of managing the notification process is on each SP, and, as mentioned, this is a way to detect the attack and limiting its effects, and not preventing it.

7. Ethical considerations and disclosure

Though the severity of the attack strongly depends on the vulnerable website, the results we presented in the previous section are quite alarming. For instance, in the case of sp5, an e-commerce shopping platform, the attacker can make purchases on behalf of the victim. In the case of Workable, one of the world's foremost hiring platforms, the attacker can recruit talent on behalf of the victim. In the case of sp8, an SP which supports digital futures trading, the attacker can sniff the trading strategies of the victim.

Of course, we made sure that our tests did not cause any harm to the tested websites. For instance, we neither injected any code in the HTTP requests nor tried to have unauthorized access to user accounts that are not under our control. All tests were performed using test accounts created on the websites by us. Since we considered only authentication and identity management processes and replayed only HTTP messages (belonging to the user accounts we created), our testing is different from that of [27]. Additionally, when we conducted further tests with the plugin, we made sure that the SSOLinking Checker did not send too many HTTP requests, to avoid a potential denial of service.

4. https://workspaceupdates.googleblog.com/2018/04/more-secure-sign-in-chrome.html

We contacted all the 17 vendors of all the vulnerable web sites through bug bounty platforms like Hackerone[5] and Bugcrowd[6] if available, otherwise trough the contact information available on the corresponding web sites.

On web sites having well-defined communication channels to report security vulnerabilities (precisely websites including Workable, Naver), we filed vulnerability reports. To contact the others, we used the information available on their web sites for general enquiry. We will update the details about the report results on the companion web site of this paper [1].

The communication with the vendors was difficult in terms of explaining them the complexity of the attack and the impact, since it involved two different vulnerabilities. We received some positive responses for our reports. For clarification, the Naver SP patched the vulnerability and paid us a bug bounty of up to $100. Our report is now featured on the Hall of Fame page[7]. Another SP, Goodreads, owned by Amazon, has confirmed the vulnerability, which has already been patched, and offered a reward of $200. Workable acknowledged the issue and offered us non-monetary rewards for our findings.

However, no information regarding these vulnerabilities is publicly available. For all other vendors, we are either waiting for the acknowledgments or working closely with them to fix the issues. This is mainly due to the fact that the experiments concluded recently and it has not been long since we reported our findings to the affected vendors. For this reason, we will use aliases instead of the real names of the SPs.

For now, 41% of vendors responded, and of those that responded, 43% acknowledged the vulnerability. The results of the disclosure procedure can be found in Table 5. We will update the details on the companion website of this paper [1].

5. https://hackerone.com/
6. https://bugcrowd.com/
7. https://bugbounty.naver.com/ko/halloffame_2022

TABLE 5: Overview of the SP disclosure procedure.

SP Name	responded	vuln confirmed	issue fixed	reward
sp1	no	-	-	-
sp2	no	-	-	-
sp3	yes	waiting	-	-
naver.com	yes	yes	-	yes
sp4	no	-	-	-
sp5	yes	waiting	-	-
goodreads.com	yes	yes	yes	yes
sp6	yes	waiting	-	-
sp7	no	-	-	-
sp8	no	-	-	-
sp9	no	-	-	-
workable.com	yes	yes	yes	no
sp10	no	-	-	-
sp11	no	-	-	-
sp12	no	-	-	-
sp13	yes	waiting	-	-
sp14	no	-	-	-
Total	**7**	**3**	**2**	**2**

8. Limitations

Though we neither present a novel CSRF attack on OAuth, nor we test a large number of SPs in our experiments, still we show that an attacker who wants to hijack SP accounts does not need to invest in identifying new CSRF attack vectors on OAuth as they can simply reuse the known, yet understudied, attack vector we highlight in our paper. Additionally, we focus on finding vulnerabilities on top-ranked SP websites to indicate that lower-ranked websites are likely more vulnerable.

The limited number of SPs targeted is also due to the lack of automation of the SP selection process. For every SP, we had to visit the website, register an account, and verify the supporting SSOLinking with the selected IdPs. We could not automate this fully because past studies show that top websites integrate automation-prevention techniques to their registration and login processes. We encountered them in the form of captchas and bot-detection libraries (as we reported in Section 5.1).

The lack of automation in detecting Authentication CSRF vulnerabilities in IdPs is another limitation. We decided not to invest in this automation because most IdPs we considered either had known and unfixed Authentication CSRF vulnerabilities reported in past studies or had a feature that could be abused for Authentication CSRF.

9. Related Work

Past research in the intersection of SSO and CSRF presents four main findings that are relevant for our research. We discuss them below.

1. Authentication CSRF vulnerabilities in IdPs: In 2008, Barth *et al.* [5] presented the lack of CSRF protection in the login form of prominent IdPs (including Google and Facebook). A variant of this attack affecting the Facebook and Twitter IdPs were discovered by Lundeen *et al.* [22] and Bansal *et al.* [4] respectively. Sudhodanan *et al.* [33] found a different way to do Authentication CSRF in Twitter IdP through the URLs embedded in the emails sent by Twitter. Calzavara *et al.* [9] presented an attack vector based on the SAML protocol to forcefully authenticate users to an attacker-controlled Google IdP account. While these studies show the lack of (or incorrect implementation of) CSRF defenses in IdPs, we show how features such as one-click login can be repurposed as Authentication CSRF attack vectors.

2. Incorrect CSRF protection in SSO callback endpoints: In 2008, Barth *et al.* [5] first presented the idea of leveraging the callback URL of SSO protocols (such as OpenID) to forcefully authenticate users to attacker-controlled accounts in SPs. They also proposed the mitigation of adding non-guessable parameters in the callback HTTP request to protect from the attack. Prominent SSO protocols such as OAuth 2.0 implements this mitigation (e.g., see Section 2.1 of [21]). Akhawe *et al.* [2] found the same attack on multiple implementations of the WebAuth SSO protocol. Sun and Beznosov [35], Bansal *et al.* [4], Shernan *et al.* [30], Yang *et al.* [37], Sudhodanan *et al.* [34], and Bonelli *et al.* [6] showed that several implementations of the OAuth protocol are also vulnerable to the same attack. When SSO protocols are leveraged for linking an IdP account to an SP account, the same vulnerability can lead to SP account hijacking. This was demonstrated by Wang *et al.* [36], Li and Mitchell [19], Yang *et al.* [37], and Sudhodanan *et al.* [34]. In this paper, we show that even when an SP protects their SSO callback URL with non-guessable tokens, the flow may still be vulnerable to CSRF. Although Bonelli *et al.* [6] explores this idea in their tests on OAuth implementations, they focus on the incorrect validation of the `state` parameter by the SP. However, our study shows that even if the `state` parameter is validated correctly by the SP, the OAuth flow is vulnerable to CSRF if the SSO initiation request is not protected.

3. Missing CSRF protection in the SSO initiation request: In 2012, Sun and Beznosov [35] presented *Force-login CSRF* attack where an adversary leverages the lack of CSRF protection in the HTTP request that initiates the SSO flow at SPs. When the victim visits an attacker-controlled website, the attacker triggers this request. If the victim is already logged-in at the IdP, the SSO flow executes seamlessly, and the victim is forcefully-logged into the SP. This attack is presented as a launchpad for mounting Post-Authentication CSRF attacks at the SP. Bansal *et al.* [4] showed that the same attack is applicable to the SP citysearch.com with Facebook as the IdP. Interestingly, in 2013, Lundeen [22] showed that the same attack can be escalated to an SP account hijack if the IdP suffers from an Authentication CSRF vulnerability and if the SSO flow is associated to linking the IdP account at the SP. This attack is widely understudied and past research has not measured the incidence of this vulnerability on top websites. We address this shortcoming by performing a tool-assisted empirical evaluation of the attack.

4. Other works related to CSRF attacks & defenses: In 2022, Khodayari and Pellegrino [18] evaluated how well the `SameSite` attribute and `StrictByDefault` policy prevent CSRF attacks, emphasizing their efficacy in securing against same-site state alterations in post-authentication. In [10], Compagna *et al.* studied the effectiveness of the `SameSite` attribute and *LaxByDefault* policy in thwarting CSRF attacks, particularly targeting same-site state changes post-authentication. Squarcina *et al.*, in [31], assessed cookie integrity risks and the effectiveness of current protections against network and same-site attackers. In 2021, Likaj *et al.* [20] emphasized the importance of CSRF defense awareness among developers, discussed implementation challenges, highlighted additional security risks, and pointed out documentation inadequacies. In [17], Khodayari and Pellegrino focuses on client-side CSRF, a novel vulnerability where adversaries manipulate client-side JavaScript to send forged HTTP requests to vulnerable sites. They also introduced JAW, a framework utilizing declarative traversals on hybrid property graphs to analyze such vulnerabilities.

10. Conclusion and Future Work

Our research addresses the security challenges posed by SSO processes, whose cross-site nature complicates significantly the CSRF protections. We focused on the SSOLinking and on an overlooked and understudied CSRF attack vector that allows an attacker to perform Account Hijack. We proposed a security testing approach to assist a tester to automatically detect CSRF vulnerabilities enabling that Account Hijack. By leveraging SSOLinking Checker—the prototype we have implemented—we performed an experimental analysis on prominent websites. The results are quite alarming: 21 out of 48 unique pairs SP-IdP suffered from SSOLinking CSRF vulnerabilities (i.e. 43.7%). On the one hand, our study emphasizes the persistent presence of CSRF vulnerabilities in SSO processes and the effectiveness of our approach to spot them. On the other hand, our work stresses the importance of raising more awareness in the community to fight these still very prevalent vulnerabilities.

As a future work, since our proposed approach—based on a declarative language to specify tests and generate Selenium scripts on-the-fly—is extremely flexible we plan to apply it to test other relevant cross-site scenarios. Additionally, it would be beneficial to understand what could be done on the IdP side to provide even better SDKs, documentations and code snippets to enable websites to securely integrate with IdPs, and ensure eradication of CSRF issues from these scenarios.

Acknowledgments

This work has been partially funded by *(i)* the European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206, *(ii)* the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union — NextGenerationEU, and *(iii)* the joint laboratory between Fondazione Bruno Kessler (FBK) and the Italian Government Printing Office and Mint, Italy. We would like to thank the anonymous reviewers for their valuable comments that help us to improve the quality of the paper, and to Amir Sharif and Paolo Prem for their support and insights.

References

[1] Supporting materials. https://st.fbk.eu/complementary/EuroSP2024.

[2] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John Mitchell, and Dawn Song. Towards a formal foundation of web security. In 2010 23rd IEEE Computer Security Foundations Symposium, pages 290–304, 2010.

[3] Elham Arshad, Michele Benolli, and Bruno Crispo. Practical attacks on Login CSRF in OAuth. Computers & Security, 121:102859, 2022.

[4] Chetan Bansal, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Sergio Maffeis. Discovering concrete attacks on website authorization by formal analysis. Journal of Computer Security, 22(4):601–657, April 2014.

[5] Adam Barth, Collin Jackson, and John C Mitchell. Robust defenses for cross-site request forgery. In Proceedings of the 15th ACM conference on Computer and communications security, pages 75–88, 2008.

[6] Michele Benolli, Seyed Ali Mirheidari, Elham Arshad, and Bruno Crispo. The full gamut of an attack: An empirical analysis of OAuth CSRF in the wild. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 21–41. Springer, 2021.

[7] Andrea Bisegna, Roberto Carbone, Ivan Martini, Valentina Odorizzi, Giulio Pellizzari, and Silvio Ranise. Micro-Id-Gym: Identity Management Workouts with Container-Based Microservices. Int. J. Inf. Secur. Cybercrime, 8(1):45–50, 2019.

[8] Andrea Bisegna, Roberto Carbone, Giulio Pellizzari, and Silvio Ranise. Micro-Id-Gym: A Flexible Tool for Pentesting Identity Management Protocols in the Wild and in the Laboratory. In International Workshop on Emerging Technologies for Authorization and Authentication, pages 71–89. Springer, 2020.

[9] Stefano Calzavara, Riccardo Focardi, Matteo Maffei, Clara Schneidewind, Marco Squarcina, and Mauro Tempesta. WPSE: Fortifying Web Protocols via Browser-Side Security Monitoring. In 27th USENIX Security Symposium (USENIX Security 18), pages 1493–1510, 2018.

[10] Luca Compagna, Hugo Jonker, Johannes Krochewski, Benjamin Krumnow, and Merve Sahin. A preliminary study on the adoption and effectiveness of SameSite cookies as a CSRF defence. In 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pages 49–59. IEEE, 2021.

[11] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1953–1970, 2020.

[12] Burp Suite Community Edition. Portswigger. https://portswigger.net/burp/communitydownload.

[13] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. O Single Sign-Off, Where Art Thou? An Empirical Analysis of Single Sign-On Account Hijacking and Session Management on the Web. In 27th USENIX Security Symposium (USENIX Security 18), pages 1475–1492, 2018.

[14] Mark Goodwin and Mike West. Same-site cookies draft-west-first-party-cookies-07. Technical report, 2016.

[15] Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and Marc Sleegers. Shepherd: a generic approach to automating website login. 2020.

[16] Florian Kerschbaum. Simple cross-site attack prevention. In 2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007, pages 464–472. IEEE, 2007.

[17] 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), pages 2525–2542, 2021.

[18] Soheil Khodayari and Giancarlo Pellegrino. The state of the same-site: Studying the usage, effectiveness, and adequacy of samesite cookies. In 2022 IEEE Symposium on Security and Privacy (SP), pages 1590–1607. IEEE, 2022.

[19] Wanpeng Li and Chris J Mitchell. Security issues in OAuth 2.0 SSO implementations. In International Conference on Information Security, pages 529–541. Springer, 2014.

[20] Xhelal Likaj, Soheil Khodayari, and Giancarlo Pellegrino. Where we stand (or fall): An analysis of CSRF defenses in web frameworks. In Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses, pages 370–385, 2021.

[21] Torsten Lodderstedt, John Bradley, Andrey Labunets, and Daniel Fett. OAuth 2.0 Security Best Current Practice. https://www.ietf.org/archive/id/draft-ietf-oauth-security-topics-21.html, 2022.

[22] Rich Lundeen. The Deputies are Still Confused. https://www.youtube.com/watch?v=_LEvuF_1O5A.

[23] Mozilla. Fetch metadata request header. https://developer.mozilla.org/en-US/docs/Glossary/Fetch_metadata_request_header.

[24] Mozilla. SameSite cookies. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite.

[25] OWASP. Cross Site Request Forgery (CSRF). https://owasp.org/www-community/attacks/csrf.

[26] OWASP. What changed from 2013 to 2017? https://owasp.org/www-project-top-ten/2017/Release_Notes.

[27] Giancarlo Pellegrino and Davide Balzarotti. Toward black-box detection of logic flaws in web applications. In NDSS, volume 14, pages 23–26, 2014.

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

[29] Open Web Application Security Project. Cross-Site Request Forgery Prevention Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

[30] Ethan Shernan, Henry Carter, Dave Tian, Patrick Traynor, and Kevin Butler. More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations. In Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148, DIMVA 2015, page 239–260, Berlin, Heidelberg, 2015. Springer-Verlag.

[31] Marco Squarcina, Pedro Adão, Lorenzo Veronese, and Matteo Maffei. Cookie Crumbles: Breaking and Fixing Web Session Integrity. In 32nd USENIX Security Symposium (USENIX Security 23), pages 5539–5556, 2023.

[32] Sristee. Getting started with Selenium: Guide to automated UI testing. https://www.divami.com/blog/selenium-guide-to-automated-ui-testing/.

[33] Avinash Sudhodanan, Alessandro Armando, Roberto Carbone, and Luca Compagna. Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications. In NDSS, 2016.

[34] 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), pages 350–365. IEEE, 2017.

[35] San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: an empirical analysis of oauth sso systems. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 378–390, 2012.

[36] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. Explicating SDKs: Uncovering assumptions underlying secure authentication and authorization. In 22nd USENIX Security Symposium (USENIX Security 13), pages 399–314, Washington, D.C., August 2013. USENIX Association.

[37] Ronghai Yang, Guanchen Li, Wing Cheong Lau, Kehuan Zhang, and Pili Hu. Model-based security testing: An empirical study on oauth 2.0 implementations. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS '16, page 651–662, New York, NY, USA, 2016. Association for Computing Machinery.