

# DISTINCT: Identity Theft using In-Browser Communications in Dual-Window Single Sign-On

Louis Jannett  
Ruhr University Bochum  
louis.jannett@rub.de

Vladislav Mladenov  
Ruhr University Bochum  
vladislav.mladenov@rub.de

Christian Mainka  
Ruhr University Bochum  
christian.mainka@rub.de

Jörg Schwenk  
Ruhr University Bochum  
joerg.schwenk@rub.de

## ABSTRACT

Single Sign-On (SSO) protocols like OAuth 2.0 and OpenID Connect 1.0 are cornerstones of modern web security, and have received much academic attention. Users sign in at a trusted Identity Provider (IdP) that subsequently allows many Service Providers (SPs) to verify the users' identities. Previous research concentrated on the standardized — called *textbook* SSO in this paper — authentication flows, which rely on HTTP redirects to transfer identity tokens between the SP and IdP. However, modern web applications like single page apps may not be able to execute the textbook flow because they lose the local state in case of HTTP redirects. By using novel browser technologies, such as *postMessage*, developers designed and implemented SSO protocols that were neither documented nor analyzed thoroughly. We call them *dual-window* SSO flows.

In this paper, we provide the first comprehensive evaluation of dual-window SSO flows. In particular, we focus on the In-Browser Communication (InBC) used to exchange authentication tokens between SPs and IdPs in iframes and popups. We automate our analysis by developing DISTINCT — a tool that dynamically analyzes the JavaScript code executing as part of the SSO flow. DISTINCT translates the flow into a sequence diagram depicting all communicating entities and their exchanged messages, highlights insecure communication channels, and quantifies novel threats in dual-window SSO flows. We found that 56% of the SPs in the Tranco top 1k list support dual-window SSO. Surprisingly, 28% of the SPs implemented dual-window SSO without using official SDKs, leading to identity theft and XSS in 31% of these self-implemented SPs.

## CCS CONCEPTS

• **Information systems** → *Web applications*; • **Security and privacy** → **Authentication**; *Authorization*; *Web application security*.

## KEYWORDS

Single Sign-On; OAuth; OpenID Connect; Web Security; Identity

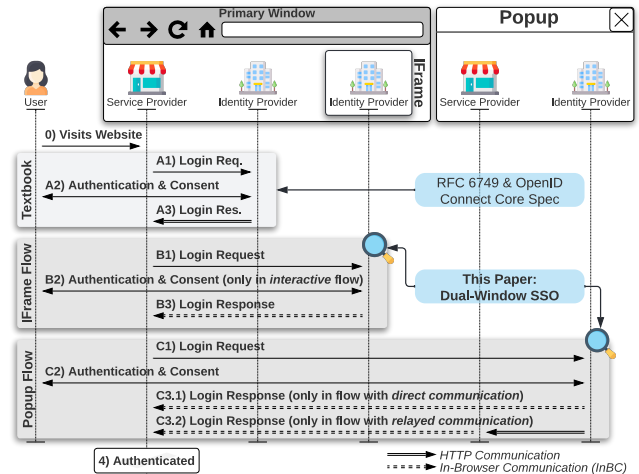
## ACM Reference Format:

Louis Jannett, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. 2022. DISTINCT: Identity Theft using In-Browser Communications in Dual-Window Single Sign-On. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*, November 7–11, 2022, Los Angeles, CA, USA. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3548606.3560692>



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA  
© 2022 Copyright held by the owner/author(s).  
ACM ISBN 978-1-4503-9450-5/22/11.  
<https://doi.org/10.1145/3548606.3560692>



**Figure 1: Textbook vs. Dual-Window SSO Authentication Flows.** Textbook SSO uses standardized HTTP-redirect-based communication to transfer the token within the login response from the IdP to the SP. In contrast, dual-window SSO uses iframe and popup flows which rely on In-Browser Communication (InBC) techniques like *postMessage* to exchange tokens between distinct windows.

## 1 INTRODUCTION

**Textbook Single Sign-On.** Single Sign-On (SSO) is a frequently used and widely prevalent user authentication scheme for websites. In 2022, David G. Balash et al. [8] showed that 86% of their recruited participants used Google's SSO service on at least one website. Roughly every fourth website in the Tranco top 1k list implements SSO to authenticate users with Apple, Facebook, and Google (cf. §6). For starting the authentication, a user clicks on the sign-in button on a Service Provider (SP) website (cf. Figure 1). Then, the SP switches the user's browser to the Identity Provider (IdP) and sends the login request. The user logs in on the IdP and provides consent that the IdP may share personal data with the SP. Finally, the IdP switches the user's browser back to the SP with the login response, which contains SSO tokens that are used for the authentication. In the *textbook SSO flow*, as defined in the OAuth 2.0 (OAuth) and OpenID Connect 1.0 (OIDC) specifications [24, 58], login requests and login responses are technically HTTP redirects. The security of this type of token exchange is well studied and already addressed by the security community [3, 5, 13, 37, 38, 39, 45, 71, 76] and standardization bodies [40, 41, 78]. However, we show that more than half of the websites with SSO deviate from these specifications and are, thereby, not covered by previous work.

**Dual-Window Single Sign-On.** Web applications are in a constant shift towards a more app-like experience. For instance, single page apps (SPAs) like Reddit change their local state (including the URL address bar) without ever reloading the browser's primary window. In these cases, textbook SSO [24, 58], which relies on HTTP redirects, is no longer viable. Thereby, website developers must execute the SSO flows in iframes or popups (cf. Figure 1). The major challenge of such flows is the token transfer from the IdP's popup / iframe to the SP's primary window. Therefore, web applications use several In-Browser Communication (InBC) techniques, such as *postMessage* and *Channel Messaging*. We call such flows *dual-window SSO flows* if they use InBC instead of HTTP communication. In this paper, we focus on the security of these techniques in the context of SSO. We investigate the token exchange, which is the most crucial part of the SSO authentication flow. If the token leaks to an attacker, an account takeover is possible.

**Monitoring In-Browser Communication.** InBC does not generate any HTTP traffic. Thus, security analysts cannot use HTTP proxies like OWASP ZAP [55] to monitor and analyze this communication. There are tools, such as PMForce [64] or Burpsuite's DOMinvasion [56], which use dynamic code execution to detect misconfigurations on the *receiving* side of *postMessage* communications. In this paper, we concentrate on the analysis of the *sending* side and consider further InBC techniques beyond *postMessage* (cf. §3.3). For this purpose, we developed a novel tool for the dynamic code analysis in web browsers: DISTINCT can record and analyze InBCs between all windows involved in the SSO flow.

**Security of In-Browser Communication.** Cross-origin InBC techniques allow windows of different origins to exchange messages. It provides a controlled relaxation of the Same Origin Policy (SOP), which protects a website from being accessed by another cross-origin site. The best-known InBC technique is *postMessage* [51]. When utilized in SSO protocols, *postMessage* enables the cross-origin communication between the SP and IdP — entirely without HTTP redirects. Like any authentication protocol that exchanges tokens, SSO sets rigorous security requirements. Although *postMessage* provides security checks to meet these demands, for instance, the confidentiality and authenticity of messages, these checks are optional and developers must implement them manually. Prior work [23, 52, 63, 64, 66] has revealed that the lack of a “security by default” approach is a substantial problem on websites, resulting in Cross-Site Scripting (XSS) and information leaks. If message confidentiality is not guaranteed in SSO, token leakages leading to identity theft are the most prominent threats.

**Research Questions.** Recent research studied the security of InBC [52, 64]. It focuses on how to protect the *recipient* against malicious data injection. We extend this work in two dimensions. First, we study the security of InBC in the context of user authentication through SSO, which automatically makes the actual content of InBC messages security-relevant. Second, we focus on the protection of the *sender* in InBCs, which has not received much academic attention yet. Thereby, we answer the following research questions:

- (1) How are dual-window SSO flows implemented?
- (2) How can we analyze dual-window SSO flows automatically?
- (3) How many websites implement dual-window SSO?
- (4) How secure are dual-window SSO implementations?

**RQ1: Systematization of InBCs in Dual-Window SSO.** In §3, we highlight the differences between textbook SSO and dual-window SSO. We systematically describe four SSO flows depending on the window's type (iframe vs. popup) and the InBC techniques (i.e., *postMessage*, *JS Navigate*). Finally, we systematize 9 different techniques used for the in-browser token exchange. To the best of our knowledge, we are the first providing such an in-depth analysis.

**RQ2: Analyzing Methodology.** Previous research elaborated on techniques to analyze textbook SSO with automated HTTP traffic inspection [37, 45, 57, 62, 76]. Since InBCs do not cause any HTTP traffic, these approaches reach their limits. We fill this gap in §5 by presenting the open-source tool DISTINCT. We use dynamic code inspection similar to Steffens et al. [64], but in contrast to them, we also monitor the sending side. We further inspect eight additional same-origin and cross-origin InBC techniques, where *postMessage* is only one particular but significant case. DISTINCT visualizes and automatically analyzes the complex SSO login flow, highlights insecure InBCs, and generates exploits.

**RQ3: Dual-Window SSO in the Wild.** Our objective is to detect, analyze, contextualize, and evaluate InBCs in SSO. We use DISTINCT to evaluate five widely deployed SSO JavaScript SDKs and the SSO implementations on the Tranco top 1k websites, of which 27% support SSO. The size of this test set is comparable to recent research on SSO (cf. §6). We show that 56% of the SSO-supporting websites use InBCs. This distribution proves the significant gap in prior work. Our results are summarized in Table 2.

**RQ4: Security Evaluation.** Security vulnerabilities in SSO implementations can have severe consequences, up to a complete impersonation of the victim at an SP. Based on the security analysis of the different dual-window SSO flows in §4, we created a requirements catalog covering security checks which need to be fulfilled for InBCs. In our evaluation (§7), we differentiate between SPs using official SDKs and SPs implementing SSO manually. As expected, we do not find any security issues in the SDK-based usage. However, we discover vulnerabilities in 31% of the websites that manually implement the InBC in their SSO flows. These vulnerabilities range from privacy disclosures to token theft resulting in unauthorized access and account takeover. We summarize our findings in Table 3.

**Contributions.** We make the following contributions:

- We are the first to systematize dual-window SSO flows in OAuth and OIDC. They deviate from the textbook SSO specification, rely on InBC instead of HTTP redirects, and were not thoroughly studied in previous research (§3).
- We show how web security threats affect the security of dual-window SSO, and we introduce attacks on dual-window SSO that have not yet been in the SSO community's focus (§4).
- We present DISTINCT, the first open-source<sup>1</sup> tool that detects and analyzes InBCs in SSO automatically. On top, DISTINCT can automatically identify vulnerable flows and provide proof of concept exploits and sequence diagrams that depict the communicating entities (§5).
- We evaluate the Tranco top 1k websites and present the landscape of implemented SSO flows. Our investigation reveals that 56% of the SPs deploy dual-window SSO (§6).

<sup>1</sup>DISTINCT's source code is available on <https://github.com/RUB-NDS/DISTINCT>. We also provide a website running a public demo on <https://distinct-ssso.com>.

- We discover vulnerabilities in 24 high-ranking websites, such as alibaba.com, aliexpress.com, and nytimes.com, and investigate the security of five popular SSO SDKs (§7).

**Responsible Disclosure.** We conducted a coordinated disclosure of the identified vulnerabilities to the respective website operators and helped in fixing them.

## 2 BACKGROUND: TEXTBOOK SSO FLOWS

The term SSO commonly refers to *textbook SSO flows* specified in [24, 58], which use HTTP-based communication techniques. In particular, textbook SSO flows are characterized by a login flow that (1) uses *URL Redirects* to exchange messages between the IdP and SP, and (2) is entirely executed within a single window, the primary window (see Figure 1). The SP sends the login request to the IdP, which returns the login response back to the SP via the user’s browser if the user provides consent. The login response contains sensitive tokens (e.g., code, id\_token, access\_token) which authenticate the user on the SP.

**HTTP-based Communication Techniques.** The general purpose of SSO is to transfer the user’s authentication on the IdP back to the SP. Its underlying challenge is to use a *communication technique* that allows transferring SSO messages from one website to a different one. In textbook SSO, the transfer is implemented using a *URL Redirect* from the IdP to the SP, along with dedicated parameters (e.g., code or id\_token). For instance, *URL Redirects* can directly change the window’s location through server-initiated redirects (e.g., HTTP/302) or using HTML (e.g., via `<meta>` or `<form>`). Although the origin of the redirect’s initiator and its receiver may differ, so that redirects can transfer SSO messages cross-origin, this technique only works in the same *window* object. As such, it is used in the textbook SSO’s primary window. Another popular HTTP-based communication technique is CORS, which is however not used in the textbook SSO flow.

## 3 DUAL-WINDOW SINGLE SIGN-ON FLOWS

In this section, we are the first to describe the *dual-window SSO flows*. These flows have essential differences compared to the textbook flows, resulting in severe impacts for SSO security and privacy. We found these flows in many modern web applications like SPAs, which do not want to lose control of the primary window by redirecting the user to its IdP. With dual-window SSO, developers can use popups or iframes to hold control of the primary window during the login process. We asked ourselves how these dual-window flows work, as they require rigorous changes to the standardized communication techniques. For instance, *URL Redirects* used by textbook SSO are no longer feasible, as they cannot communicate across multiple windows. To the best of our knowledge, we are the first to provide a systematic study of dual-window SSO flows like the *popup flow* (§3.1) and *iframe flow* (§3.2).

**Textbook vs. Dual-Window SSO.** The textbook protocol flow has been the primary target in previous works. Researchers and penetration testers identified several security issues in its implementations [3, 5, 11, 16, 17, 34, 35, 36, 37, 39, 42, 45, 61, 68, 70, 71, 73, 75, 76] and specifications [13, 15, 25, 44, 53], ranging from simple replay attacks to complex authentication bypasses. Therefore, they inspected the HTTP traffic. Instead, we found that dual-window SSO

Paper	Technology	Scope	Iframe / Popup	Actively Used <sup>1</sup>	Autom. PoC	Threat Model	Analysis Methodology
[23]	FBC, GFC	IdP	✓/✗	✗	✗	WA	Manual source code analysis, reverse engineering
[71]	FBC	IdP	✗/✓	✗	✗	WA	Manual analysis of Flash-based InBC
[14]	BrowserID	IdP & SP	✓/✓	✗	✗	WA+	Formal model, manual protocol analysis
[12]	SPRESSO	IdP & SP	✓/✓	✗	✗	WA+	New SSO protocol, formal model
[19]	OAuth, OIDC	IdP	✓/✗	✓	✗	JS Ex.	Manual evaluation of <i>post-Message</i> prevalence in SSO
This Paper	OAuth, OIDC	IdP & SP	✓/✓	✓	✓	WA	Automated capturing, analysis, visualization, and exploitation of InBCs

<sup>1</sup> Custom or obsolete SSO protocols are considered as *not actively used*.

FBC: Facebook Connect GFC: Google Friend Connect

WA: Standard Web Attacker Model [1]

JS Ex.: Requires untrusted JavaScript execution

WA+: Extended Web Attacker Model

**Table 1: Prior work on SSO protocols relying on InBC. In this paper, we (1) investigate SP implementations of dual-window SSO, (2) introduce the iframe flow with user interaction and two popup flows that are based on OAuth and OIDC, and (3) open-source a tool that automatically captures, analyzes, and exploits InBCs in dual-window SSO.**

uses In-Browser Communication (InBC) techniques to exchange SSO messages between two windows, for example, the primary window and a login popup. These messages are not visible in the HTTP traffic and thus, were missed in their analyses. Our study complements these works by highlighting the risks of using InBCs in dual-window web SSO.

**Initial Studies on Dual-Window SSO.** Beyond textbook SSO, there have been initial studies on SSO protocols executed in iframes and popups [12, 14, 19, 23, 71]. Table 1 shows an overview of them. We provide a more thorough comparison to related works in §8. To summarize, our work differs in (1) considering state-of-the-art technologies; (2) additionally investigating custom SP implementations; (3) introducing new flows that have not yet been in the SSO community’s focus; (4) presuming the web attacker with low exploit requirements; and (5) automatically analyzing and exploiting InBCs in dual-window SSO.

### 3.1 Popup-Based Single Sign-On Flows

Figure 1 depicts the popup flow. The primary window opens the IdP’s popup once the user starts the login on the SP’s website. The user authenticates on the IdP and consents the SP’s access permission to resources. Then, the popup flow terminates and the popup receives the login response from the IdP. Since the user wants to authenticate to the SP, which runs in the primary window, the popup must return the established authentication. This return relies on InBC between the popup and primary window. In practice, there are two variants to perform the InBC in the popup flow.

(1) *Direct Popup Flow:* The IdP uses InBC to directly communicate with the SP, returning the login response from the popup to the primary window. This variant does not require the SP to act as a relay within the popup. Expired specification drafts [26, 30, 74] already describe this flow on an abstract level.

(2) *Relayed Popup Flow:* The IdP uses a *URL Redirect*, which is based on HTTP communication, inside the popup to return the

login response to the SP. Then, the SP loaded in the popup forwards the login response from the popup back to the primary window using InBC. This flow is neither standardized nor has been the subject of any prior research.

### 3.2 IFrame-Based Single Sign-On Flows

Figure 1 depicts the iframe flow. It works similar to its popup equivalent, but it is executed within an iframe rather than a popup. Therefore, the login request is opened within an iframe embedded on the SP's website. The user optionally authenticates and consents within the iframe, without ever leaving the website running in the primary window. Finally, the iframe communicates with its parent frame to return the login response. The iframe-based flow can significantly improve the login experience on websites. We distinguish between two variants, each offering a different login experience.

(1) *Seamless IFrame Flow*: If the user is logged in at the IdP and has granted consent, for instance, an account on the SP, the iframe flow can provide a *seamless* login experience. When the iframe receives a login request, the IdP silently authenticates the user and immediately returns the login response to the SP (i.e., B2 in Figure 1 is skipped). In this case, the SP can make the iframe invisible such that the user profits from a seamless, automatic sign-in.

(2) *Interactive IFrame Flow*: If either the authentication at the IdP or the consent is missing, the user needs to *interact* with the IdP's iframe, i.e., click the consent button. The required user interaction with the iframe has a significant downside. It enables UI redressing and clickjacking attacks [24, §4.4.1.9], in which attackers trick victims into interacting with the iframe without noticing it, for instance, to obtain consent and tokens by fraud. The community knows about these attacks and mitigates them with countermeasures like *X-Frame-Options* or the *frame-ancestors CSP* [40, § 4.15]. These countermeasures are disruptive because they prohibit the consent page from being embedded. However, we found that the *Intersection Observer v2 API*<sup>2</sup> [65] allows an embedded iframe to enforce its integrity and mitigate clickjacking risks in a non-destructive way. The iframe's JavaScript code can query the API and check whether it is utterly unoccluded by other content. If the API returns a positive indicator, it guarantees the iframe's unobstructed visibility to the user.

### 3.3 In-Browser Communication Techniques

Dual-window SSO uses InBC techniques like *postMessage* and in-browser storage containers to transfer and store authentication tokens. Various InBC techniques have been in the scope of different research [4, 79], which we used as a starting point for our analysis. We extend, systematize, and describe all InBC methods that can be used to transfer authentication tokens.

#### Terminology.

(1) The *initiator* starts the communication. This is any *window* object<sup>3</sup> in the DOM, like the primary window, an iframe, or a popup. The initiator's security context is bound to its origin<sup>4</sup> (protocol, domain, port). In SSO, this is either the IdP's or SP's origin.

(2) The *receiver* is the communication partner of the initiator. Similarly, it can be any *window* object. The receiver's security context is likewise bound to its origin. It can either have the same origin as the initiator's origin, or it can have a different one, which we call the cross-origin case.

(3) InBCs between an initiator and receiver use a dedicated *technique*. Commonly used techniques are *postMessage*, *Channel Messaging*, *JS Direct Access*, *JS Storage*, and *JS Navigate* via the location's fragment. The choice of a technique affects the communication's security, and they can work for same-origin or cross-origin communications. Cross-origin InBC techniques are exciting for SSO since the IdP wants to send tokens to authenticate the user on the cross-origin SP. Same-origin InBC techniques can be used only if the initiator and the receiver share the same origin. It is notable that all cross-origin techniques also work in the same-origin case.

(4) Techniques imply transferring an *In-Browser Message (InBM)* from the initiator to the receiver. In the context of SSO, the IdPs and SPs commonly exchange tokens. In this paper, we concentrate on InBMs that must be kept secure and prevented from being received by malicious actors, for instance, tokens and user data.

**Security-Relevant InBC Techniques.** InBC techniques must fulfill two requirements to be security-relevant: (1) The InBC technique must work for cross-origin communications. For example, the attacker's website resides on a different origin than the attacked site (cf. §4.1). Note that if a website uses same-origin InBC techniques, the SOP inherently prevents other, potentially malicious origins from sending and receiving InBMs. Thereby, we consider all same-origin InBC techniques secure. (2) The InBC technique must be capable of sending InBMs containing data like strings or objects. Otherwise, an attacker can neither receive confidential InBMs like authentication tokens nor send maliciously crafted InBMs to vulnerable sinks. Two InBC techniques — *Cross-Origin Web Messaging* and *JS Navigate* — fulfill both requirements to be relevant for the security of dual-window SSO.

(1) *Cross-Origin Web Messaging*: We found that two JavaScript methods can be grouped into this technique. It includes the *postMessage* API [51] and *Channel Messaging* API [47]. These APIs were designed for cross-origin communications between two windows. An initiator can send a *postMessage* to a receiver by calling the `postMessage(message, receiverOrigin)` method that each *window* provides. The receiver can listen on the *"message"* event to retrieve the InBM. *Channel Messaging* works similarly. By invoking `new MessageChannel()`, the initiator creates a channel with two entangled ports. One port is transferred to the receiver (e.g., using *postMessage*) and thus provides a two-way communication channel.

(2) *JS Navigate*: This technique allows the initiator to navigate the receiver to an arbitrary URL using JavaScript. In particular, the SOP allows cross-origin write access to the *window.location* property [50]. By setting the *window.location* property, InBMs can be attached to the URL using query or fragment parameters. In contrast to *URL Redirects*, browsers can use JavaScript to navigate different *window* objects (e.g., popups and iframes), making it applicable for dual-window SSO flows. Additionally, *JS Navigate* behaves special if it only changes the fragment of the URL. In that case, the receiver window does not reload but is notified about a change in its fragment with the *"hashchange"* event.

<sup>2</sup>The Intersection Observer v2 API is available in Chromium-based browsers, like Edge  $\geq v79$ , Chrome  $\geq v74$ , and Opera  $\geq v62$  (<https://caniuse.com/intersectionobserver-v2>).

<sup>3</sup><https://html.spec.whatwg.org/multipage/window-object.html>

<sup>4</sup><https://html.spec.whatwg.org/multipage/origin.html>



**Safe InBC Techniques.** The cross-origin *JS Properties* technique allows the initiator to query certain DOM properties of the receiver. For instance, the SOP allows cross-origin read access to the `frames.length` and `window.closed` properties [27, 29, 67]. In light of same-origin techniques, we identified *Same-Origin Web Messaging*, which incorporates the *Broadcast Messaging API* [46] and *JS Custom Events* [48]. Both APIs work similar to *postMessage* but for same-origin communications only. With *JS Direct Access*, (1) the receiver can expose a JavaScript callback function that the initiator invokes, or (2) the initiator can set properties in the receiver's DOM. By using *JS Storage*, the initiator can store InBMs in storage containers like `localStorage`, `sessionStorage`, `IndexedDB`, and cookies, and the receiver can retrieve them. *JS Reload* lets the initiator tell the receiver to reload its page, i.e., using its `location.reload()` function.

## 4 SECURITY IN DUAL-WINDOW SSO

We present the new attack surface that dual-window SSO introduces on top of the textbook security considerations [40, 41]. Therefore, we use our study of dual-window SSO flows in §3 as a basis.

### 4.1 Threat Model

We rely on the web attacker model proposed by Akhawe, Barth, Lam, Mitchell, and Song [1]. In the following, we reveal an abstract overview of the web attacker's capabilities, approach, and goals.

**Web Attacker.** The attacker hosts a malicious website, for example, `attacker.com`, and can lure the victim to visit it. In contrast to previous research [19, 20, 21], the attacker cannot execute JavaScript on the attacked website, for instance, `sp.com`. We further exclude software or hardware deficiencies, and we assume standard-conform, non-compromised web browsers and secure TLS.

**Victim.** The victim uses SSO to authenticate on an honest SP's website (i.e., `sp.com`). We assume that the victim is logged in at the IdP (i.e., `idp.com`) and provided consent, i.e., allowed the IdP to share personal data with the honest SP. Thus, the SSO authentication takes place seamlessly and without any user interactions. To that end, the attacks introduced in this section do not require the victim to interact with the IdP or SP.

**Approach and Goals.** We assume the SSO flow to use InBC techniques to exchange InBMs between the SP and IdP. The attacker first embeds an SP's or IdP's endpoint into the malicious website, either via iframes or popups. The victim loads the malicious website, which exploits insufficient security checks of InBCs to (1) *send messages*, breaking the message authenticity, or (2) *receive messages*, breaking the message confidentiality. For example, `attacker.com` communicates with `sp.com` or `idp.com`. We consider an attack successful if the malicious website can receive a security-relevant SSO message. Similarly, the attack succeeds if the malicious site can send such an SSO message that is accepted by the receiver.

### 4.2 Security of InBC Techniques

**Security of Cross-Origin Web Messaging.** *PostMessage* and *Channel Messaging* are the most convenient techniques for cross-origin InBC and particularly designed by browsers for this purpose. Both APIs can exchange arbitrary InBMs, which may include confidential data like tokens.

(1) *PostMessage*: While *postMessage* allows the enforcement of confidentiality and authenticity of InBMs, these guarantees are not provided by default. To achieve confidentiality, for instance, preventing other origins from receiving the *postMessage*, the initiator must explicitly set the receiver's origin in the *postMessage* call (*receiver origin check*):

```
receiverWindow.postMessage(InBM, "https://receiver.com")
```

To ensure authenticity, for instance, assuring that the received *postMessage* was actually sent by the expected initiator, the receiver must explicitly verify the initiator's origin in the *postMessage* handler (*initiator origin check*):

```
receiverWindow.onmessage = (InBM) => {  
  if (InBM.origin !== "https://initiator.com") return  
}
```

(2) *Channel Messaging*: The initiator creates a new channel with two entangled ports and transfers one port via *postMessage* to the receiver. All InBMs sent to or received from the channel are confidential and authentic because only the two origins in possession of a port may communicate via the channel. Thus, the security of the *Channel Messaging* API relies on the confidentiality and authenticity of the *postMessage* that transfers the port.

**Security of JS Navigate.** The SOP allows each window to set the `window.location` property of other cross-origin windows. For example, the initiator may set the receiver's location as follows:

```
receiverWindow.location = "https://receiver.com/recv?InBM#InBM"
```

The InBM can be included within the URL's query or fragment. The receiver extracts the InBM by reading its `window.location` property. The URL specified by the initiator inherently guarantees confidentiality because the InBM is explicitly sent to `receiver.com`. However, the receiver cannot determine and thus verify the initiator setting the `location` property, which breaks the InBM's authenticity. In practice, well-known Cross-Site Request Forgery (CSRF) protection mechanism like CSRF tokens compensate this loss of authenticity. Their concept is simple yet effective: (1) The initiator and receiver share a common secret.<sup>5</sup> (2) Whenever the initiator sets the receiver's location, it includes the secret in the InBM. (3) The receiver extracts the secret from the InBM and validates it. If the validation succeeds, the receiver can reason that the InBM was actually sent by the trusted initiator.

### 4.3 Attacks in Dual-Window Single Sign-On

We identified three attacks in dual-window SSO that have not yet been in the SSO community's focus. The first two attacks target the confidentiality; the third targets the authenticity of SSO messages.

**Security-Relevant SSO Messages.** The most crucial message during the SSO login process is the login response. It contains confidential tokens granting access to the victim's identity and resources. Hence, its confidentiality must be protected such that it may only be received by the authorized SP, which issued the related login request and received the victim's consent. Similarly, only the IdP that received the login request is allowed to issue the related login response to the SP. In summary, the attacker aims to (1) receive a login response from a flow that the victim started on the SP, and (2) send a maliciously crafted login response to the SP.

<sup>5</sup>In SSO, the `state` query parameter [24, § 10.12] serves as a secret that is shared between the IdP and SP to ensure that location writes are authentic.

### Attack Impact: Loss of Confidentiality.

(1) *Token Leak (TL)*: If authentication tokens leak to the attacker, identity theft, account takeovers, and unprivileged resource accesses are possible. Depending on the token redemption's stealthiness, the victim may not notice the attacker's account access.

(2) *Privacy Leak (PL)*: If private data leaks to the attacker, the victim's identity may be revealed. Likewise, other user-identifiable information like the victim's email, address, and profile picture may leak. We consider token leaks enabling account takeovers as inherent privacy leaks.

### Attack Impact: Loss of Authenticity.

(1) *Cross-Site Account Signin (XSAS)*: The victim is logged into the attacker's account, allowing the attacker to track the victim's activities. For example, if the victim uploads a file, it is uploaded into the attacker's account, and the attacker may retrieve it.

(2) *Cross-Site Account Linking (XSAL)*: The victim is already logged in on the SP's website with credentials (i.e., username and password). Later, the victim decides to connect the account on the SP to an IdP. If the attacker sends a malicious login response, the victim's account on the SP is linked to the attacker's account on the IdP. The victim can still use the credentials to log into the SP. On top, the attacker can use the SP's SSO service to log into the victim's account (i.e., account takeover).

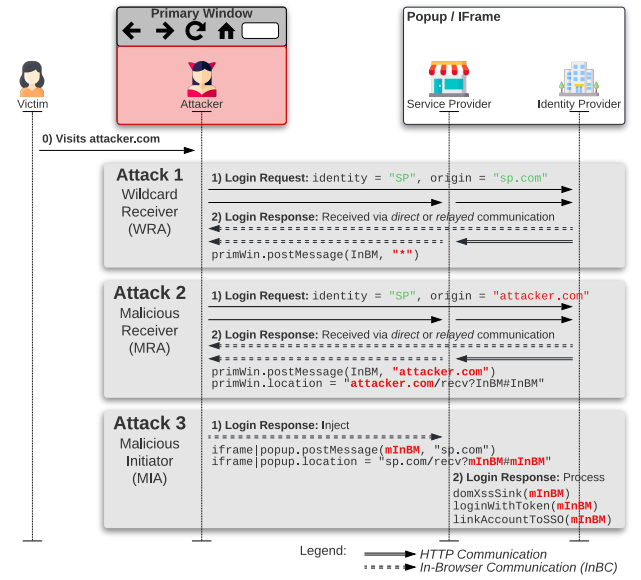
(3) *DOM-based Cross-Site Scripting (DOM-XSS)*: The SP processes the login response within JavaScript sinks that enable DOM-XSS [33], like `eval()` and `innerHTML`. The attacker sends a malicious login response to exploit the sinks and gain DOM-XSS on the SP.

**Attack 1: Wildcard Receiver Attack (WRA)**. Figure 2 depicts a malicious website exploiting the WRA to receive a login response. The attack works as follows: The victim visits the malicious website, which imitates the SP on which the victim has an account with SSO. The malicious site starts the SSO login flow that is normally started by the SP. Therefore, the attacker sends the SP's login request to the IdP or to the SP, which forwards it to the IdP. Both parties successfully verify that the given origin, which should receive the login response (i.e., `sp.com`), is authentic and trusted. Thus, the IdP generates the login response, which is returned to the malicious website. We distinguish between two cases:

(1) *Direct Communication*: The IdP directly returns the login response to the primary window. However, the IdP fails to protect its confidentiality, as *Cross-Origin Web Messaging* is used insecurely in this context. The *postMessage wildcard "\*"* allows all origins to receive the login response. Hence, the IdP leaks the login response, which likely contains tokens or privacy-sensitive data, to the attacker's site. Direct communications are used by both iframe flows and the direct popup flow.

(2) *Relayed Communication*: In this case, the IdP first uses a *URL Redirect* to return the login response to the SP running within the popup. Second, the SP uses insecure InBC to return the login response from the popup to the primary window. Thus, the SP leaks the login response to the attacker's site. Relayed communication is used exclusively in the relayed popup flow.

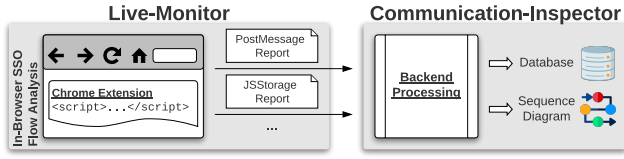
**Attack 2: Malicious Receiver Attack (MRA)**. Figure 2 illustrates the MRA, which is closely related to the WRA but with an additional twist. The InBC techniques *postMessage* and *js Navigate* are used securely, as they explicitly include the receiver's origin to protect the confidentiality. However, the trustworthiness of the



**Figure 2: Attacks in Dual-Window Single Sign-On.** In the *Wildcard Receiver Attack (WRA)*, the receiver's origin of the login response is not specified. In the *Malicious Receiver Attack (MRA)*, the receiver's origin of the login response is not properly validated. Thus, authentication tokens may leak. In the *Malicious Initiator Attack (MIA)*, the SP does not validate the initiator's origin of the login response. Thus, malicious sites may inject a malicious login response or JavaScript code.

receiver's origin is not or not properly validated by the SP or IdP. Thus, the attacker replaces the SP's origin (`sp.com`) with the malicious origin (`attacker.com`). The login response is finally sent to the non-verified origin `attacker.com`, running in the primary window.

**Attack 3: Malicious Initiator Attack (MIA)**. Figure 2 shows the MIA. The attacker's website imitates the IdP to send a login response to the SP. In the MIA, the SP does not or not properly verify the authenticity of the login response. Best practices to counter the attack is to perform a string comparison on the initiator's origin, for example, to verify `InBM.origin ==? "https://idp.com"` in *postMessage*. Examples of vulnerable checks include comparisons with overly lax regular expressions or string comparison methods such as `startsWith` and `includes`. Our attacks assume the attacker to have an account on the IdP and to execute the SSO flow on the SP. Thus, the attacker receives a "malicious" login response from the IdP. If the victim visits the malicious site, the attacker sends the *malicious login response* to the SP which unleashes the following impact: (1) The SP uses the login response to log the victim into the attacker's account (cf. XSAS, §4.3). (2) If the victim is already logged in on the SP, the SP links the victim's account to the attacker's account on the IdP (cf. XSAL, §4.3). (3) The SP passes the login response into dangerous JavaScript sinks like `eval`. The attacker adapts the login response to exploit the sink and gain XSS on the SP (cf. DOM-XSS, §4.3).



**Figure 3: DISTINCT’s Design and Architecture.** The Chrome extension (LIVE-MONITOR) sends all InBC techniques with their InBMs in reports to the backend (COMMUNICATION-INSPECTOR), which post-processes them and, inter alia, outputs a sequence diagram highlighting threats.

## 5 DISTINCT: DYNAMIC IN-BROWSER SINGLE SIGN-ON TRACER INSPECTING NOVEL COMMUNICATION TECHNIQUES

Prior research assessed SSO implementations by analyzing the HTTP traffic. This approach misses crucial SSO messages sent via InBC techniques at runtime. We are the first filling this gap by providing DISTINCT as an open-source contribution to SSO implementers, researchers, and pentesters, raising awareness for the security pitfalls of dual-window SSO. DISTINCT automates the (1) capturing, (2) detection, (3) visualization, (4) security analysis, and (5) exploitation of dual-window SSO flows with InBCs.

### 5.1 Design and Architecture

DISTINCT is designed to monitor and analyze the InBC techniques sending InBMs across windows at runtime. Therefore, DISTINCT executes two components: LIVE-MONITOR and COMMUNICATION-INSPECTOR (see Figure 3).

**Components.** (1) LIVE-MONITOR is a Chrome extension that runs in the context of all browser windows (i.e., primary windows, iframes, and popups), injects a monitoring script into each page, and gets access to the entire site, including its DOM and all APIs. LIVE-MONITOR transfers the collected runtime data in reports to a backend, the COMMUNICATION-INSPECTOR. For instance, a report contains detailed information about a specific *postMessage* that is sent or URL that is assigned to a *window*’s location.href. (2) COMMUNICATION-INSPECTOR is a Python backend that gathers, stores, inspects, and visualizes the reports generated by LIVE-MONITOR. Therefore, it serves an API and a web application. The API first receives the reports from LIVE-MONITOR, and then triggers their post-processing. During the post-processing, COMMUNICATION-INSPECTOR automatically performs the flow visualization, security analysis, and exploitation. The web application is DISTINCT’s primary control interface allowing analysts to start a new analysis and inspect the results in a searchable table and sequence diagram.

**Workflow.** We designed DISTINCT as a containerized Docker application that is straightforward to install and operate via a single web interface. DISTINCT can be optionally preconfigured with credentials for the IdPs to automate their authentication and consent prompts. To start a new analysis of an SSO flow on a target website, the analyst has to manually submit its URL in DISTINCT’s web interface. Then, the web interface shows the Chromium browser,

which is connected via NoVNC<sup>6</sup>. The browser automatically loads LIVE-MONITOR and navigates to the targeted website. Next, the analyst has to manually locate and click the website’s SSO login button to start the SSO authentication flow. If the IdP’s credentials were preconfigured, the authentication and consent step on the IdP is automated. Otherwise, the analyst has to manually authenticate on the IdP and click the consent button. During the entire login flow, LIVE-MONITOR automatically reports all InBCs to the COMMUNICATION-INSPECTOR, which starts the automated post-processing. Finally, the analyst can manually inspect the analysis results in the web interface. It displays the login flow’s sequence diagram showing the detected security threats. If threats are detected, the analyst can trigger the automated exploitation, manually verify its success, and perform minor changes if necessary.

**Analysis Approaches.** To analyze SSO-related InBMs, we need to detect the invocations of APIs that are used by the InBC techniques in JavaScript. Therefore, we examined the static and dynamic JavaScript analysis approaches.

*Static analysis* investigates the JavaScript code before the browser executes it. Although this is a reasonable approach in numerous instances, it has its downsides: (1) We cannot determine the script’s execution time and order. (2) We cannot detect the window in which the script is executed. (3) JavaScript minification and obfuscation complicates the analysis. These reasons impede the SSO flow reconstruction so we decided to use a dynamic analysis.

*Dynamic analysis* monitors the code executing at runtime to detect API calls. As a side effect, the runtime analysis solves challenges induced by JavaScript minification and obfuscation. It also provides inherent information about the time of code execution, and we can determine the window that runs it. Thus, we consider the dynamic analysis as a robust approach for capturing the API calls of InBC techniques in their execution time and window.

### 5.2 LIVE-MONITOR

LIVE-MONITOR performs an automatic runtime analysis to answer the following questions: (1) Which textbook or dual-window SSO flow is started? (2) Does the SP use an SDK or manual implementation? (3) How are the InBC techniques used?

**Automated Flow Detection.** LIVE-MONITOR detects SSO messages like the login request and login response based on known recognition patterns from [43]. For instance, if it observes a request to `accounts.google.com` that contains well-known SSO parameters [24, 58] like `client_id`, it marks this request as the *login request* and sets the flow’s IdP to *Google*. Similarly, it detects a *login response* if it contains well-known tokens [24, 58] like `code`, `access_token`, or `id_token`. For the first time, LIVE-MONITOR recognizes dual-window SSO flows by mapping the SSO messages to their windows, iframes, and popups. Thereby, LIVE-MONITOR can automatically discern textbook flows from dual-window flows.

**Automated SDK Detection.** To automatically detect SSO SDKs, we extended LIVE-MONITOR with new recognition patterns since they were not covered in [43]. For this purpose, we executed the SSO logins with the SDKs once, analyzed the messages, and extracted recognition patterns for each SDK. We noticed that the

<sup>6</sup> NoVNC allows us to display the graphical desktop of a remote computer in a web interface. More information is available on <https://novnc.com>.



SDKs' login requests involved SDK-specific parameters in addition to the well-known SSO parameters, for instance, `frame_id` (Apple), `channel_url` (Facebook), and `ux_mode` (Google). We implemented them as rules to detect whether a website uses an SSO SDK.

**Automated InBC Detection.** LIVE-MONITOR can identify a window's hierarchy with a recursive algorithm. Without this hierarchy, it would not be able to track the InBCs between iframes and popups. We express a window's hierarchy as its relation to the primary window. For example, `top.frames[1].popups[0]` denotes the first popup opened by the primary window's second iframe. Based on our preliminary studies in §3.3, we need a systematic approach capable to (1) automatically detect the usage of InBC techniques throughout the login flow, and (2) determine their related InBMs, for instance, the exchanged data. The challenge is the engineering and implementation of reliable InBC observers. We identified six main approaches for monitoring: *Passive Monitoring*, *Event Listener*, *Getter & Setter*, *Value / Function Wrapper*, *Proxies*, and *Prototypes*. LIVE-MONITOR modifies JavaScript objects, functions, and APIs to capture variable values and messages within the browser. As a result, it can automatically determine InBCs and extract their initiators, receivers, and messages. As an example, we provide further insights on how we monitored *postMessage* in the following.

**Automated *postMessage* Detection.** LIVE-MONITOR detects the *postMessage* receivers similar to Meiser et al. [52] and Steffens et al. [64]. For instance, it hooks the `addEventListener()` function on each *window* and wraps *postMessages* in JavaScript Proxy objects [49] to track accesses on their properties. Thus, LIVE-MONITOR detects and traces the initiator origin checks via the *postMessages*' origin property, and reports all property accesses to the COMMUNICATION-INSPECTOR. The detection of *postMessage* initiators was more challenging since no dynamic, in-browser detection systems have been implemented yet. LIVE-MONITOR similarly hooks the `postMessage()` function on each *window* to get access to its arguments, for instance, the receiver origin check. However, these hooks override the browser's native *postMessage* function such that the *postMessage* receiver loses access to its initiator. To restore this connection, LIVE-MONITOR sets a breakpoint on the receiver and uses the Chrome DevTools protocol [7] to run debugging actions automatically. If a *postMessage* hits the breakpoint on the receiver, the debugger automatically attaches and steps back to the initiator in the execution trace. Once there, it sets the original origin and source properties of the *postMessage* to restore the reference to the initiator. To the best of our knowledge, this approach detects all *postMessage* initiators. In theory, receivers could identify proxied *postMessages* by checking its inheritance from the `MessageEvent` interface. In practice, we could not find any receivers implementing this check.

### 5.3 COMMUNICATION-INSPECTOR

The COMMUNICATION-INSPECTOR implements: (1) the automated threat detection, (2) the automated flow visualization & threat highlighting, and (3) the automated Proof of Concept (PoC) generation.

**Automated Threat Detection.** The threat detection engine that is built into the COMMUNICATION-INSPECTOR has access to the entire database of InBCs and InBMs. It operates passively on that database and avoids the modification of communications to

not interfere with the complex session dependencies in SSO. We implemented three threat detection approaches:

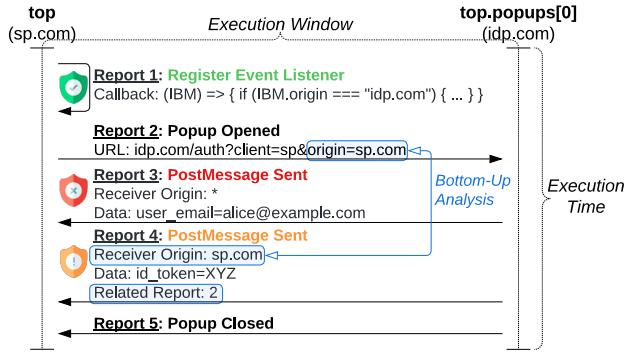
(1) *Wildcard Receiver Patterns:* The *postMessage* wildcard "\*" is harmful as information leaks to the attacker. Since we store all *postMessage* calls with their arguments in the database, we can highlight all leaky *postMessages* that are using the "\*". For instance, the third report in the sequence diagram (cf. Figure 4) uses the wildcard origin leaking the user's email, and it is marked accordingly.

(2) *Initiator Verification Patterns:* The origin property of a *postMessage* provides a reliable pattern to detect vulnerable initiator origin checks. LIVE-MONITOR tracks all accesses to this property during runtime and provides them to the COMMUNICATION-INSPECTOR for further analyses. Prior work [52, 63, 64] performed in-depth assessments of the *postMessage* initiator verification, so we could safely adopt their origin-related patterns: If the origin is not accessed, no check is performed at all. Invocations of deficient or error-prone string functions on the origin are also a reliable pattern. For instance, `startsWith()`, `endsWith()`, `indexOf()`, `includes()`, `match()`, and `search()` are best known to be used in insecure initiator origin checks, which we mark accordingly. Based on Terjanq's XSS-challenge [69], we implemented a verification pattern to identify the vulnerable `window.origin` check, which has not been considered in the context of SSO before. In addition to prior work [52, 63, 64], we consider *JS Navigate*. Since it does not provide a mechanism to verify its initiator, we mark all of them as potentially forgeable.

(3) *Bottom-Up Analysis:* If no wildcards are used, it is difficult to determine if the attacker can manipulate the receiver's origin. To solve this problem, we implemented a bottom-up analysis into the COMMUNICATION-INSPECTOR. We start the analysis with the receiver's origin specified in the *postMessage* or *JS Navigate*. Then, we search backward in our database for user-manipulable parameters that match this origin, and we highlight all occurrences. The analyst manually tests whether an active manipulation of the detected parameter influences the receiver's origin in the *postMessage* or *JS Navigate*. In that case, the parameter is not accurately validated. Thus, the MRA is possible. Figure 4 shows an example of the bottom-up analysis. The *postMessage*'s receiver origin is found to be also included in the login request's origin query parameter. The IdP must validate the login request and only allow origins that are whitelisted by the SP.

**Automated Flow Visualization.** DISTINCT's COMMUNICATION-INSPECTOR automatically generates sequence diagrams to visualize the dual-window SSO flows. In addition, it highlights security-relevant messages and recognized threats. Figure 4 depicts a sequence diagram of a popup flow. First, the SP in the primary window registers an event listener, which safely validates the initiators of received InBMs. Second, the SP opens a popup, navigates to the IdP, and sends the login request. Third, the IdP sends two *postMessages* back to the SP before it closes the popup: (1) The first *postMessage* contains the user's email but does not set a receiver origin. Thus, it is susceptible to the WRA enabling a privacy leak. (2) The second *postMessage* contains an authentication token and sets its receiver origin to the SP's origin. The bottom-up analysis reveals that the *postMessage*'s receiver origin matches the login request's origin parameter. The analyst must manually verify whether an active manipulation of this parameter is reflected in the *postMessage*'s receiver origin.





**Figure 4: Sequence Diagram of a Dual-Window SSO Flow.** The diagram allows the analyst to determine (1) when the InBCs occurred, and (2) in which windows they were involved. **Confirmed** or **potential** threats are highlighted.

**Automated PoC Generation.** We implemented an automated PoC exploitation engine into the COMMUNICATION-INSPECTOR. The engine relies on DISTINCT’s database and threat detection engine. It automatically builds, outputs, and deploys a PoC website as HTML file with inline JavaScript. We provide exemplary PoCs exploiting our vulnerable test service in the demo running on [distinct-ss0.com](https://distinct-ss0.com).

For exploiting WRA and MRA, the PoC website simulates the SP, initiates the SSO flow, and, if the attack is successful, receives the login response that the IdP issued for the SP. Therefore, the engine proceeds as follows: (1) It determines the window in which the login request is sent. (2) It identifies the *initial URL* that is first opened in this window. This step compensates any redirections executed before the login request is sent to the IdP. (3) The PoC website opens / embeds the initial URL in a popup / iframe, starting the SSO flow. (4) To fully simulate the SP, the PoC site sends all captured InBCs to the popup / iframe that were formerly sent by the SP to them. In some cases, this is required to trigger the token issuance. (5) The PoC website receives the login response in its registered receiver if its confidentiality is broken.

For exploiting MIA, the PoC website simulates the IdP, embeds or opens the SP’s login page, and sends the login response to the SP. If the attack is successful, the SP starts processing the login response. Therefore, the engine proceeds as follows: (1) It determines all active receivers and the windows in which they are registered. (2) It embeds or opens the receivers’ windows in iframes or popups. (3) It sends all InBCs to the receivers’ windows that were sent by the former initiators to them. (4) If the receivers fail to verify their authenticity, they start to process the InBCs.

## 5.4 Limitations

**Automated Vulnerability Exploitation.** DISTINCT can exploit the WRA. However, the exploitation of the MRA and MIA is limited and may require adjustments on the PoC. Steffens et al. [64] unveiled the significant engineering to automatically exploit insecure receivers for XSS. For us, the SP’s non-uniform parameters and session dependencies add another layer of complexity. Their automated detection and active testing is open for future research.

**Automated Vulnerability Verification.** Not every insecure usage of InBC leads to account takeover or other vulnerabilities. Thus, DISTINCT implements patterns to filter security-relevant messages, but experts must manually verify the results. During our research, we determined that SPs use many custom parameters to exchange encoded tokens or user data. It is hard to determine if an insecurely exchanged InBM contains security-relevant data.

**SSO Login Automation.** We manually clicked the SSO buttons on the SPs for reliability and technical reasons. We still evaluated two strategies to fully-automate the logins: (1) extend DISTINCT to automatically find and execute SSO logins, or (2) take advantage of prior works. Previous works have been able to conduct large-scale web and SSO security studies post-authentication [9, 17, 28, 54, 61, 76]. They all required significant engineering to achieve an unsatisfactory detection rate, which by far surpasses the scope and focus of this paper. We failed to find a tool that can serve our requirements: (1) open-source availability, (2) support for state-of-the-art SSO logins in today’s landscape, and (3) support for sign-ins with Apple, Google, and Facebook. [17, 54, 61] only detect SSO logins, but do not execute them. [28, 76] only support Facebook, while [9] additionally supports Google. However, [9, 17, 28] are not publicly available, and our contact attempts were unsuccessful. Overall, non-maintained tools (i.e., [76] was last updated 7 years ago) cannot execute SSO as it’s found in today’s, fast-paced SSO landscape. That said, a generic open-source framework for automated logins with state-of-the-art SSO is desirable and left for future work.

**Improve Code Coverage.** Although DISTINCT’s dynamic analysis suites well for our empirical evaluation (cf. §4), it may not cover all execution paths of a website. For instance, certain user interactions with the SP may invoke vulnerable API calls, which DISTINCT cannot detect if they are not executed. DISTINCT can only report on the code that is executed in the SSO flow.

**Further Identity Providers.** We focused on the top three IdPs and found that their SDKs are secure (cf. §4). However, there are numerous other IdPs on the Internet. Whether they provide official SDKs and whether they are as secure as Apple’s, Facebook’s, and Google’s SDKs is left open. DISTINCT might also be useful for companies that deploy their own IdP with dual-window SSO flows.

## 6 EVALUATION: DUAL-WINDOW SINGLE SIGN-ON LANDSCAPE

In this section, we present the distribution of dual-window SSO in the wild. We split our landscape analysis into two parts: (1) *buttons* starting the textbook or dual-window SSO flows, and (2) *websites* supporting textbook or dual-window SSO flows.

**Methodology.** In 2021, Morkonda et al. [54] showed that Apple, Facebook, and Google are the three most predominantly supported IdPs on the Alexa Top 500. Thus, we focused on these three IdPs. We studied their implementation guidelines and SSO SDKs [2, 10, 18] to determine the flows they support. To select a representative list of SPs, we used the top 1k domains in the Tranco list<sup>7</sup> [32]. Compared to related work [17, 39, 54, 70, 75], the number of analyzed websites in this paper is similar or larger. For each website, we started an analysis run in DISTINCT’s web interface (cf. “Workflow” in §5.1). We manually clicked on the SSO buttons of each website.

<sup>7</sup>Generated on 22 July 2021. Available at <https://tranco-list.eu/list/ZGXG>.

Meanwhile, LIVE-MONITOR automatically traced the login flows. LIVE-MONITOR’s automated flow and SDK detection finally reveals implementation details regarding the SSO landscape, which we summarize in Table 2.

**SSO Detection Rate.** The false positive rate (FPR) for LIVE-MONITOR’s SSO and SDK detection is 0%, respectively the true positive rate (TPR) is 100%. Since we manually clicked on the SSO buttons on all 1k websites, we confirmed that LIVE-MONITOR correctly identified and classified all textbook and dual-window SSO flows. Since the login request is uniquely detectable, this accuracy is reasonable. However, we noticed that Google published a new SDK during DISTINCT’s development. Thus, we extended LIVE-MONITOR with a new recognition pattern to detect it.

## 6.1 Part 1: Buttons starting SSO Flows

We used DISTINCT’s LIVE-MONITOR to count the number of SSO buttons on the Tranco top 1k websites (see part 1 in Table 2). Note that there is not a one-to-one connection between websites and buttons. For example, a single website can include multiple SSO buttons, each using a different IdP or flow. We identified a total of 683 SSO buttons: 414 buttons started dual-window flows (61%); 269 buttons started textbook flows (39%).

**Manual Integrations.** All IdPs provide public SSO APIs for manual SSO integrations. These APIs allow websites to implement the textbook flow or relayed popup flow, which use *URL Redirects* to return the login response to the SP in the primary window or popup, respectively. For example, we found 57 buttons starting the textbook flow with Apple manually. Interestingly, 153 buttons started the relayed popup flow. However, SDKs do not support this flow, thus, it is always implemented manually.

**SDKs.** Apple and Facebook each provide a single SDK: Sign in with Apple (SiwA<sub>🍏</sub>) and Facebook Login (FL<sub>f</sub>). Google offers three SDKs: Sign in with Google (SiwG<sub>🍏</sub>), Google Sign-In (GSI<sub>🍏</sub>), and Google One Tap (GOT<sub>🍏</sub>). Unsurprisingly, buttons provided by Google SDKs are the most prevalent ( $3 + 114 + 74 = 191$ ). SiwA<sub>🍏</sub> starts the direct popup flow if its “use popup” option is set. This was the case for 18 buttons on the Tranco top 1k websites. Otherwise, it initiates the textbook flow, for which we found 12 buttons. FL<sub>f</sub> starts the direct popup flow if the user has to authenticate or consent. Otherwise, it issues a CORS request to fetch tokens from the backend. We identified 53 FL<sub>f</sub> buttons across the landscape. SiwG<sub>🍏</sub> is the successor of GSI<sub>🍏</sub> and starts the textbook or direct popup flow, based on the “ux mode” option. We found only 3 buttons provided by this SDK, which is the by far least distributed SDK in our landscape. GSI<sub>🍏</sub> is the predecessor of SiwG<sub>🍏</sub> and now discontinued as legacy SDK, but as we show, still the most distributed SDK. If GSI<sub>🍏</sub> requires user interaction, it starts the direct popup flow. Otherwise, it launches the seamless iframe flow to automatically log the user into the SP. If GSI<sub>🍏</sub> is in its seamless configuration, the user must not explicitly click a button — thus, no button is displayed. However, we still consider this as a “simulated” click, which the SDK automatically performs. This example illustrates how a single SDK on one website can start multiple flows, depending on the given circumstances. GOT<sub>🍏</sub> initiates the seamless iframe flow, unless the user has not yet consented. Similar to GSI<sub>🍏</sub>, GOT<sub>🍏</sub> only “simulates” the button click in its seamless variant. To the best of

our knowledge, GOT<sub>🍏</sub> is the first SDK implementing the interactive iframe flow with the Intersection Observer v2 API (cf. §3.2). As for non-supported browsers, GOT<sub>🍏</sub> falls back to the popup flow.

## 6.2 Part 2: Websites with SSO Flows

We wanted to break down the *counting of buttons* in the left part of Table 2 to the *number of websites*. Out of the Tranco top 1k, we identified 273 websites providing SSO logins with at least one of the three IdPs. For the first time, Table 2 proves that dual-window SSO exceeds textbook SSO nowadays. From 273 websites, 153 implement dual-window SSO (56%), surpassing textbook SSO implemented by 134 websites. This distribution elucidates that previous research focusing on textbook SSO implementations missed crucial InBCs and InBMs in 56% of the most popular websites with SSO. Notably, every second website with dual-window SSO uses manual integrations (77 of 153, 50%). This insight strengthens the significance of our research targeting the security of dual-window SSO.

**Manual Integrations.** Table 2 suggests that more than twice as many websites prefer manual SSO integrations over SDKs. Manual integrations primarily use textbook SSO on 127 websites. Yet, 77 sites manually integrate dual-window SSO with the relayed popup flow. Interestingly, previous works did entirely miss this flow, leaving a considerable gap in the security research on SSO. We prove the threat as we discover severe vulnerabilities in this flow (see §7).



**SDKs.** Table 2 proves that websites primarily use SDKs to run dual-window flows (92). Only 13 websites use SDKs configured to run the textbook flow. We found that the direct popup flow (76) and seamless iframe flow (77) are equally distributed. Yet, 37 websites implement the seamless iframe flow with GOT<sub>🍏</sub>.

## 7 EVALUATION: DUAL-WINDOW SINGLE SIGN-ON SECURITY

We used DISTINCT to conduct a security analysis on the Tranco top 1k SSO landscape. Our results show that 31% of websites with manual integrations of dual-window SSO fail to implement InBCs securely. Even worse, we could confirm account takeovers, severe privacy leaks, and XSS on prominent websites like alibaba.com, aliexpress.com, and nytimes.com.

**Methodology.** In our landscape analysis, we executed and traced the dual-window SSO flows on 153 distinct websites by clicking on 414 buttons (cf. Table 2). For our security analysis, we manually investigated the sequence diagrams of these flows to confirm the detected threats. In specific, we used the COMMUNICATION-INSPECTOR’s security indicators as a starting point for our analyses. For the WRA, an indicator refers to a leaky InBM. We have to manually confirm whether the InBM carries tokens or private data that are relevant for an attacker. For the MRA, an indicator denotes a parameter that we have to test actively. For the MIA, an indicator references a receiver that does not properly validate the authenticity of InBMs. If the COMMUNICATION-INSPECTOR detects an indicator, we trigger the automatic PoC generation and deployment. In the victim’s browser, we manually verify whether the PoC could receive the leaky InBM or successfully send an InBM to a vulnerable receiver. In some cases, we had to manually make minor changes to the PoC like changing specific parameters or choosing a particular domain name for the web server.

		Part 1: Buttons starting SSO Flows									Part 2: Websites with SSO Flows (websites can have multiple buttons)			
		Manual 🍏	Manual 📘	Manual 🟢	SiwA 🍏	FL 📘	SiwG 🟢	GSi 🟢	GOT 🟢	Σ		Manual	SDK	Manual or SDK
📖 Textbook SSO		57	90	109	12	✗	1	✗	✗	269	≥1x 📖	127	13	134
Dual-Window SSO	Popup Flow													
	Relayed	30	59	64	✗	✗	✗	✗	✗	153		77	✗	77
	Direct	✗	✗	✗	18	53	2	57	0	130		✗	76	76
	IFrame Flow													
	Seamless	✗	✗	✗	✗	✗	✗	57*	37*	94		✗	77	77
	Interactive	✗	✗	✗	✗	✗	✗	✗	37	37		✗	37	37
Σ		30	59	64	18	53	2	114	74	414	≥1x 📖	77	92	153
📖 + 📖		87	149	173	30	53	3	114	74	683	📖 or 📖	200	97	273

Table 2: The dual-window SSO Landscape. In sum, 273 of the Tranco top 1k websites (27%) support SSO logins. We manually started 683 SSO flows on 273 websites and traced the entire authentication with DISTINCT. DISTINCT found that 153 of 273 websites (56%) implement at least one dual-window  SSO flow, superseding the textbook  distribution (134 sites). Moreover, twice as many websites prefer manual integrations over the use of SDKs. We mark flows that are not supported by IdPs as not applicable (X). Otherwise, we denote the number of buttons (part 1) or websites (part 2) that use this flow. The seamless iframe flow does not require the user to click the button but instead automatically “simulates” the click.

















































































































































































































































































































































































Rank	SP	IdP	SSOaaS	Vulnerability (§4.3)			Impact (§4.3)					Redeem Token	Automatic PoC	Manual Verify	Responsible Disclosure			
				WRA	MRA	MIA	TL	PL	XSAS	XSAL	XSS							
51	nytimes.com	  	–															
102	aliexpress.com	 	–															
141	aparat.com		–															
156	cnet.com	 CBS																
167	alibaba.com	 	–															
206	npr.org	  	Janrain															
246	independent.co.uk		GIGYA															
247	roblox.com	 GIGYA																
265	latimes.com	  	–															
317	cbsnews.com	 	CBS															
327	marriott.com	 GIGYA																
330	digg.com		–															
364	ups.com	  	GIGYA															
366	coursera.org		–															
463	cbc.ca	  	LoginRadius															
479	zdnet.com	 CBS																
643	mirror.co.uk	 	LoginRadius															
685	notion.so		–															
697	ria.ru	  	–															
801	seekingalpha.com		–															
812	ozon.ru	 –																
824	express.co.uk	 	LoginRadius															

Table 3: The dual-window SSO Security Evaluation. We found that 24 of 77 websites (31%) with manual dual-window SSO integrations are vulnerable to the Wildcard Receiver Attack (WRA), Malicious Receiver Attack (MRA), and Malicious Initiator Attack (MIA). The great majority of them (23) fail to protect the confidentiality of InBMs, resulting in Token Leaks (TLs) and Privacy Leaks (PLs) with confirmed account takeovers on 17 sites. Yet, 6 sites fail to verify the authenticity of InBMs, enabling Cross-Site Account Signin (XSAS), Cross-Site Account Linking (XSAL), and XSS in 5, 3, and 2 cases, respectively.



## 7.1 Insecurity of Manual Integrations

In Table 3, we summarize the vulnerabilities in the SSO landscape. We found 49 of 77 websites (64%) with manual integrations of dual-window SSO using a security-relevant InBC technique. Remarkably, almost every second site (24 of 49, 49%) failed to implement them securely. The remaining 28 sites (36%) used safe, same-origin InBCs. Notably, 11 websites suffered from flaws in 4 SSO as a Service (SSOaaS) providers. They offer an intermediate brokerage service between the SPs and IdPs. In such cases, the SPs and IdPs are relaying the login requests and login responses through the broker. One advantage is that SPs can support multiple IdPs even though they implement only one broker. We manually verified each issue and found that in 10 cases, the automatic PoC works out of the box. The remaining PoCs required minor adjustments, which we summarize in this section. Further, we could redeem the leaked tokens on 17 sites, confirming account takeovers. We reported all vulnerabilities as part of our responsible disclosure process and see them being acknowledged, fixed, and thankfully rewarded.

**Wildcard Receiver Attack.** Surprisingly, 12 websites sent tokens or private data in *postMessages* without specifying a receiver origin. Prominent examples were alibaba.com, aliexpress.com, and latimes.com. Leaked data commonly included the user's id, name, email, and profile picture (i.e., coursera.org, ria.ru, seekingalpha.com). We found digg.com leaking a user's profile information, such as subscriptions. On notion.so, Google's entire login response was leaked. Others like alibaba.com, aliexpress.com, aparat.com, ozon.ru, and vnexpress.net leaked opaque bearer authentication tokens. Surprisingly, technologyreview.com leaked a "state key" and latimes.com leaked a "signon token", which we were unable to redeem. The automated PoC could trivially exploit the WRA on 11 of 12 websites. Our investigations on aliexpress.com revealed that the URL starting SSO contained a boolean reload parameter. If set to `true`, the token was sent in a secure *JS Navigate* to the token redemption endpoint. If set to `false`, a *postMessage* was leaked which however only contained an irrelevant URL and not the token. For leaking the token, we had to manually apply JavaScript-encoding to the `reload=false` parameter in the PoC.

**Malicious Receiver Attack.** A total of 12 websites failed to verify the receiver's origin. For instance, if the login popup of alibaba.com received a "ping" message via *postMessage*, it responded to its unverified initiator with a *postMessage* containing the token. The ria.ru site was similarly affected, but instead of an authentication token, the user's profile data and a CSRF token were leaked. In both cases, the PoC worked out of the box as it fully emulates all *postMessages*, including the "ping" *postMessages*. The MRA also affected 11 websites integrating the brokering services. To exploit CBS Corporation (CBS), we had to embed an additional "proxy iframe" on the PoC website. We set the iframe's unverified `xdm_e` query parameter to the PoC's origin. The login popup securely sends the token to the iframe's callback. However, the iframe uses *postMessage* to relay the token to the PoC website. It sets the receiver origin to the polluted, unverified `xdm_e` parameter. To exploit Janrain by Akamai (Janrain), we had to set the unverified `xdReceiver` query parameter, which is contained in the URL starting the SSO, to the PoC's origin. We proceeded similar with the `apikey` and `callback` parameters of LoginRadius. GIGYA

by SAP (GIGYA) allowed us to append a malicious origin to the trusted receiver's origin via the URL's `user:pwd@host` component (i.e., `sp.com@attacker.com`).

**Malicious Initiator Attack.** Compared to the WRA and MRA, we only found MIAs on 6 websites. These results are in line with Steffens et al.'s [64] large-scale analyses of *postMessage* receivers. They similarly showed that the majority of the initiator origin checks are secure. Still, nytimes.com redirects its users once the SSO flow terminates. Therefore, a *postMessage* containing the redirection URL is sent to the receiver, which assigns it to its `window.location` without verifying the initiator's origin. In the PoC, we replaced the honest redirection URL with a `javascript:alert(1)` URL to trigger the XSS. Similarly, alibaba.com assigns a received token URL to an iframe for establishing the user's session. They did not validate the initiator's origin but checked whether the lower-cased URL contains `"javascript:"`. In the PoC, we replaced the honest token URL with the attacker's token URL ( $\rightarrow$  XSAS) or a `javascript:alert(1)` URL ( $\rightarrow$  XSS) that bypasses the check. LoginRadius and aparat.com both verify the initiator's origin of the login response using the `indexOf()` and `includes()` functions, respectively. We deployed the PoC to a domain like `trusted.com.attacker.com`, which effectively bypasses both methods. LoginRadius also features an account linking process, thus, XSALs are feasible.

## 7.2 Security of SDKs

DISTINCT could not find issues with the SDKs' InBCs techniques. All SDKs properly implement the required security checks. We expected this robust security level from SDKs, since Apple, Facebook, and Google probably have the required know-how to implement InBC securely. Yet, FL<sub>f</sub> exhibited a major deficiency in one of its security checks in the past [59]. It used the regular expression `^https://.*facebook.com$` to whitelist initiators. However, `attackerfacebook.com` is a valid initiator that bypasses this check.

## 8 RELATED WORK

**Textbook Single Sign-On Security.** A large and growing body of literature has investigated the security of web SSO in textbook implementations [3, 5, 11, 16, 17, 34, 35, 36, 37, 39, 42, 45, 61, 68, 70, 71, 73, 75, 76] and specifications [13, 15, 25, 44, 53]. There is also a considerable amount of literature on SSO in mobile applications [6, 60, 70, 72, 77]. The potential of SSO vulnerability scanners performing automated security assessments of SSO implementations has been in the scope of [3, 37, 71, 73, 75, 76, 77]. Thereby, threats like token and privacy leaks, XSAS ("session swapping"), and XSAL have been gaining much attention. However, prior tools and analyses of textbook SSO primarily inspected the HTTP traffic. We show that they missed crucial SSO messages exchanged via InBCs and prove the existence of new threats in dual-window SSO by considering IdPs and, for the first time, SPs.

**Dual-Window Single Sign-On Security.** We identified prior work that investigated SSO protocols involving iframes, popups, and InBCs [12, 14, 19, 23, 71]. They are summarized in Table 1. We discuss and compare them aspect-wise in the following.

(1) *Technology and Usage.* Hanna et al. [23] reverse engineered the nowadays obsolete, proprietary SSO protocols Facebook Connect (FBC) and Google Friend Connect (GFC). Wang et al. [71]

examined FBC. Fett et al. [14] investigated BrowserID. This protocol was deprecated in 2016 due to a low adoption rate [31]. Later, they proposed a new SSO scheme called SPRESSO [12]. Unfortunately, it did not reach real-world adoption until today. Similar to Guan et al. [19], we consider OAuth and OIDC, which are widely deployed and in the scope of *current* SSO research [16, 22, 39, 54].

(2) *Scope*. Hanna et al. [23] and Guan et al. [19] studied *postMessage* in SSO implementations of IdPs. Wang et al. [71] focused on the IdP’s Flash-based InBC. Fett et al. [12, 14] formally examined the underlying protocols that use *postMessage*. We are the first to investigate *SP implementations of dual-window SSO*. Our results show that SPs miss crucial security checks (cf. §7.1), whereas IdPs protect themselves (cf. §7.2). We consider *further InBC techniques* like *Channel Messsaging*, *JS Navigate*, and *same-origin techniques*.

(3) *IFrame / Popup*. The protocols and flows studied in [12, 14, 19, 23, 71] use iframes and popups. However, they deviate from the state-of-the-art protocols and dual-window SSO flows described in this paper. For instance, BrowserID [14] and SPRESSO [12] require both, iframes *and* popups. In contrast, for OAuth and OIDC, only popups (§3.1) *or* iframes (§3.2) are required. Guan et al. [19] depicts the iframe flow *without* user interaction, which we describe in §3.2. We are the first to introduce the *iframe flow with user interaction* and both *popup flows* that are based on OAuth and OIDC.

(4) *Threat Model*. Fett et al. [12, 14] proposed an advanced web attacker model, which, inter alia, considers new browser technologies. They used their model to analyze BrowserID and SPRESSO. The “DangerNeighbor” threat [19, 20, 21] uses a strong attacker model that presumes the attacker to have JavaScript execution on the SP, for example, XSS. This assumption inherently breaks the SOP and all SSO protocols relying on InBCs. Similar to Hanna et al. [23] and Wang et al. [71], we used the standard *web attacker model* [1] allowing us to exploit our findings in Guan et al.’s [19, 20, 21] threat model, but not vice versa.

(5) *Analysis Methodology and Findings*. Hanna et al. [23] manually analyzed and reverse engineered the source code of the proprietary protocols FBC and GFC. They manually verified the exploitability of their findings, for instance, XSS, privacy leaks, and identity theft. Wang et al. [71] manually discovered that FBC also uses Adobe Flash for InBC and applied a tweak to send the secret token to a malicious, cross-origin site. Guan et al. [19] manually evaluated the real-world prevalence of *postMessage* in SSO. They did not search for missed *postMessage* checks but uncovered identity theft and privacy leaks if the attacker has JavaScript execution on the SP. Fett et al. [12, 14] carried out manual protocol analyses of BrowserID and SPRESSO. They formally proved the security and privacy of SPRESSO but found BrowserID to be vulnerable to identity theft and token injection. We instead used DISTINCT for *automated security analyses* of dual-window SSO *implementations* and *automated PoC generation*.

**PostMessage Security.** Son et al. [63] showed that *postMessage* receivers do not verify the initiator origin safely. Stock et al. [66] identified that 26% of the Alexa top 500 sites receiving *postMessages* did not verify their initiators. Steffens et al. [64] extended the observations and proposed an automated framework that systematically analyzes and exploits *postMessage* receivers at scale. Meiser et al. [52] analyzed *postMessage* receivers to build an interconnected graph of their trust relations. We, inter alia, study *postMessage*

receivers in SSO and discuss their impact on the user’s account integrity that goes beyond DOM-XSS.

Barth et al. [4] secured the confidentiality of the *postMessage* API. Stock et al. [66] showed that nearly half of their sites used *postMessage* with a wildcard. Sudhodanan et al. [67] associated *postMessage* with XS-Leaks, putting forth the observation that initiators may leak data. We connect these observations to SSO and demonstrate that leaky *postMessages* enable serious identity thefts.

## 9 CONCLUSION

This paper presents the first empirical analysis of OAuth and OIDC using dual-window SSO flows to identify threats that In-Browser Communications impose. We shed light on dual-window SSO using iframes and popups, which discern them from standardized textbook flows. Our landscape evaluation proves the prevalence of dual-window SSO: 56% of the Tranco top 1k websites with SSO implement dual-window flows, while 49% implement textbook flows. Surprisingly, every second website with dual-window SSO did not use secure SDKs, but implemented dual-window flows manually.

**Root Causes.** We could identify three primary security threats in dual-window SSO, which affected 31% of the evaluated sites with custom integrations. We responsibly reported all issues, and the developers reacted thankfully, resolving them. We can trace back all threats to two main causes regarding the unsafe use of InBCs.

(1) The InBC initiator does not enforce the message’s receiver *when sending* the message. We have seen two cases: either the initiator does not specify the receiver at all (using a wildcard → §4.3 WRA) or the initiator simply reflects the receiver’s origin from a previous message (→ §4.3 MRA). Interestingly, attackers can abuse this flaw either on the IdP-side (→ §3.1 direct popup-flow) or on the SP-side (→ §3.1 relayed popup-flow).

(2) The InBC receiver does not verify the message’s initiator *when receiving* the message (→ §4.3 MIA). Although this case was considered in web-security [4, 66, 67], we could still identify this flaw as a severe threat for dual-window SSO.

**Lessons Learned.** The lessons learned from our investigation can be summarized as follows:

(1) Developers implement custom dual-window flows but they seem to overlook that SDKs implement dual-window flows securely. If possible, we recommend the use of safe, same-origin InBC techniques, like *JS Direct Access* and *JS Storage*.

(2) The SSO community should standardize dual-window flows relying on InBC and clearly address the security threats described in this paper. Recent attempts have been published in drafts [26, 30, 74], but we see none of them being actively worked on.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers and shepherd for their valuable feedback. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972. Louis Jannett was supported by the German Federal Ministry of Economics and Technology (BMWi) project “Industrie 4.0 Recht-Testbed” (13I40V002C).

## REFERENCES

- [1] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John Mitchell, and Dawn Song. 2010. Towards a Formal Foundation of Web Security. In *2010 23rd IEEE*

- Computer Security Foundations Symposium*. IEEE, Edinburgh, United Kingdom, (July 2010), 290–304. ISBN: 978-1-4244-7510-0. doi: 10.1109/CSF.2010.27.
- [2] Apple Inc. 2022. Sign in with Apple | Developer Documentation. Retrieved 08/29/2022 from <https://developer.apple.com/sign-in-with-apple/>.
  - [3] Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, Jin Song Dong, and B Guangdong. 2013. AUTHSCAN: Automatic Extraction of Web Authentication Protocols from Implementations. In *Network and Distributed System Security Symposium (NDSS)*. Adam Barth, Collin Jackson, and John C. Mitchell. 2009. Securing Frame Communication in Browsers. *Communications of the ACM*, 52, 6, (June 2009), 83–91. ISSN: 0001-0782, 1557-7317. doi: 10.1145/1516046.1516066.
  - [5] Michele Benolli, Seyed Ali Mirheidari, Elham Arshad, and Bruno Crispo. 2021. The Full Gamut of an Attack: An Empirical Analysis of OAuth CSRF in the Wild. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Volume 12756 LNCS. Springer International Publishing, 21–41. ISBN: 9783030808242. doi: 10.1007/978-3-030-80825-9\_2.
  - [6] Eric Y. Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. 2014. OAuth Demystified for Mobile Application Developers. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, Scottsdale Arizona USA, (November 3, 2014), 892–903. ISBN: 978-1-4503-2957-6. doi: 10.1145/2660267.2660323.
  - [7] 2022. Chrome DevTools Protocol | Documentation. Retrieved 08/01/2022 from <https://chromedevtools.github.io/devtools-protocol/>.
  - [8] David G. Balash, Xiaoyuan Wu, Miles Grant, Irwin Reyes, and Adam J. Aviv. 2022. Security and Privacy Perceptions of Third-Party Application Access for Google Accounts. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, (August 2022), 3397–3414. ISBN: 978-1-939133-31-1.
  - [9] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. 2020. The Cookie Hunter: Automated Black-Box Auditing for Web Authentication and Authorization Flaws. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. Association for Computing Machinery, Virtual Event, USA, 1953–1970. ISBN: 9781450370899. doi: 10.1145/3372297.3417869.
  - [10] Facebook Inc. 2022. Facebook Login | Developer Documentation. Retrieved 08/29/2022 from <https://developers.facebook.com/docs/facebook-login>.
  - [11] Shehroze Farooqi, Fareed Zaffar, Nektarios Leontiadis, and Zubair Shafiq. 2017. Measuring and Mitigating OAuth Access Token Abuse by Collusion Networks. In *Internet Measurement Conference (IMC '17)*. Association for Computing Machinery, New York, NY, USA, (November 1, 2017), 355–368. ISBN: 978-1-4503-5118-8. doi: 10.1145/3131365.3131404.
  - [12] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2015. SPRESSO: A Secure, Privacy-Respecting Single Sign-On System for the Web. In *nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. ACM Press, Denver, Colorado, USA, 1358–1369. ISBN: 978-1-4503-3832-5. doi: 10.1145/2810103.2813726.
  - [13] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2016. A Comprehensive Formal Security Analysis of OAuth 2.0. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Association for Computing Machinery, Vienna, Austria, 1204–1215. ISBN: 9781450341394. doi: 10.1145/2976749.2978385.
  - [14] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2014. An Expressive Model for the Web Infrastructure: Definition and Application to the Browser ID SSO System. In *2014 IEEE Symposium on Security and Privacy*. IEEE, San Jose, CA, (May 2014), 673–688. ISBN: 978-1-4799-4686-0. doi: 10.1109/SP.2014.49.
  - [15] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2017. The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 189–202. doi: 10.1109/CSF.2017.20.
  - [16] M. Ghasemisharif, C. Kanich, and J. Polakis. 2022. Towards Automated Auditing for Account and Session Management Flaws in Single Sign-On Deployments. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, (May 2022), 1524–1524. doi: 10.1109/SP46214.2022.00095.
  - [17] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. 2018. 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)*. USENIX Association, Baltimore, MD, (August 2018), 1475–1492. ISBN: 978-1-939133-04-5.
  - [18] Google LLC. 2022. Google Identity | Developer Documentation. Retrieved 08/29/2022 from <https://developers.google.com/identity>.
  - [19] C. Guan, Y. Li, and K. Sun. 2017. Your Neighbors are Listening: Evaluating Post-Message Use in OAuth. In *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*. (August 2017), 210–211. doi: 10.1109/PAC.2017.30.
  - [20] Chong Guan, Kun Sun, Lingguang Lei, Pingjian Wang, Yuewu Wang, and Wei Chen. 2018. DangerNeighbor Attack: Information Leakage via postMessage Mechanism in HTML5. *Computers & Security*, 80, (July 28, 2018), 291–305. ISSN: 01674048. doi: 10.1016/j.cose.2018.09.010.
  - [21] Chong Guan, Kun Sun, Zhan Wang, and WenTao Zhu. 2016. Privacy Breach by Exploiting postMessage in HTML5: Identification, Evaluation, and Countermeasure. In *th ACM on Asia Conference on Computer and Communications Security*. ACM, Xi'an China, (May 30, 2016), 629–640. ISBN: 978-1-4503-4233-9. doi: 10.1145/2897845.2897901.
  - [22] Sven Hammann, Ralf Sasse, and David Basin. 2020. Privacy-Preserving OpenID Connect. In *th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20)*. Association for Computing Machinery, Taipei, Taiwan, 277–289. ISBN: 9781450367509. doi: 10.1145/3320269.3384724.
  - [23] Steve Hanna, Eui Chul, Richard Shin, Devdatta Akhawe, Arman Boehm, Prateek Saxena, and Dawn Song. 2010. The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives. *csberkeley.edu*, (January 2010).
  - [24] Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749. (October 2012). doi: 10.17487/RFC6749. <https://rfc-editor.org/rfc/rfc6749.txt>.
  - [25] Pili Hu, Ronghai Yang, Yue Li, and Wing Cheong Lau. 2014. Application Impersonation: Problems of OAuth and API Design in Online Social Networks. In *Second Edition of the ACM Conference on Online Social Networks - COSN '14*. ACM Press, Dublin, Ireland, 271–278. ISBN: 978-1-4503-3198-2. doi: 10.1145/2660460.2660463.
  - [26] Jacob Ideskog and Travis Spencer. 2021. OAuth 2.0 Assisted Token. Internet-Draft draft-ideskog-assisted-token-05. Internet Engineering Task Force, (March 8, 2021). 20 pages. <https://datatracker.ietf.org/doc/html/draft-ideskog-assisted-token-05>.
  - [27] Artur Janc and Mike West. 2020. Oh, the Places You'll Go! Finding Our Way Back from the Web Platform's Ill-conceived Jaunts. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, IEEE, (September 2020), 673–680. doi: 10.1109/eurospw51379.2020.00096.
  - [28] Hugo Jonker, Jelmer Kalkman, Benjamin Krumnow, Marc Slegers, and Alan Verresen. 2018. Shepherd: Enabling Automatic and Large-Scale Login Security Studies. *CoRR*, abs/1808.00840. arXiv: 1808.00840.
  - [29] Lukas Knittel, Christian Mainka, Marcus Niemietz, Dominik Trevor Noß, and Jörg Schwenk. 2021. XSinator.Com: From a Formal Model to the Automatic Evaluation of Cross-Site Leaks in Web Browsers. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, Virtual Event, Republic of Korea, 1771–1788. ISBN: 9781450384544. doi: 10.1145/3460120.3484739.
  - [30] G. Kong, N. Agarwal, and W. Denniss. 2015. OAuth 2.0 IDP-IFrame-Based Implicit Flow. Internet-Draft draft-guibinkong-oauth-idp-iframe-00. Internet Engineering Task Force, (November 21, 2015). 21 pages. <http://lists.openid.net/pipermail/openid-specs-ab/Week-of-Mon-20151116/005865.html>.
  - [31] Frederic Lardinois. 2014. Mozilla stops developing its persona sign-in system due to low adoption. (March 2014). Retrieved 08/01/2022 from <https://techcrunch.com/2014/03/08/mozilla-stops-developing-its-persona-sign-in-system-because-of-low-adoption/>.
  - [32] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *th Annual Network and Distributed System Security Symposium (NDSS 2019)*. (February 2019). doi: 10.14722/ndss.2019.23386.
  - [33] Sebastian Lekies, Ben Stock, and Martin Johns. 2013. 25 Million Flows Later - Large-scale Detection of DOM-based XSS. In *ACM SIGSAC Conference on Computer & Communications Security - CCS '13*. ACM Press, Berlin, Germany, 1193–1204. ISBN: 978-1-4503-2477-9. doi: 10.1145/2508859.2516703.
  - [34] Wanpeng Li and Chris J Mitchell. 2016. Analysing the Security of Google's implementation of OpenID Connect. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 357–376.
  - [35] Wanpeng Li and Chris J. Mitchell. 2014. Security Issues in OAuth 2.0 SSO Implementations. In *Information Security (Lecture Notes in Computer Science)*. Sherman S. M. Chow, Jan Camenisch, Lucas C. K. Hui, and Siu Ming Yiu, editors. Volume 8783. Springer International Publishing, Cham, 529–541. ISBN: 978-3-319-13257-0. doi: 10.1007/978-3-319-13257-0\_34.
  - [36] Wanpeng Li, Chris J. Mitchell, and Thomas Chen. 2018. Mitigating CSRF attacks on OAuth 2.0 Systems. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 1–5. doi: 10.1109/PST.2018.8514180.
  - [37] Wanpeng Li, Chris J. Mitchell, and Thomas Chen. 2019. OAuthGuard: Protecting User Security and Privacy with OAuth 2.0 and OpenID Connect. In *th ACM Workshop on Security Standardisation Research Workshop*. Association for Computing Machinery, (November 2019), 35–44. ISBN: 9781450368322. doi: 10.1145/3338500.3360331. arXiv: 1901.08960.
  - [38] Wanpeng Li, Chris J. Mitchell, and Thomas Chen. 2018. Your Code Is My Code: Exploiting a Common Weakness in OAuth 2.0 Implementations. In *Conference: 2018 16th Annual Conference on Privacy, Security and Trust (PST)*. Volume 11286 LNCS. Springer Verlag, 24–41. ISBN: 9783030032500. doi: 10.1007/978-3-030-03251-7\_3.
  - [39] Guannan Liu, Xing Gao, and Haining Wang. 2021. An Investigation of Identity-Account Inconsistency in Single Sign-On. In *Web Conference 2021*. ACM, Ljubljana Slovenia, (April 19, 2021), 105–117. ISBN: 978-1-4503-8312-7. doi: 10.1145/3442381.3450085.



- [40] Torsten Lodderstedt, John Bradley, Andrey Labunets, and Daniel Fett. 2021. OAuth 2.0 Security Best Current Practice. Internet-Draft draft-ietf-oauth-security-topics-18. Internet Engineering Task Force, (April 13, 2021). 53 pages. <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-18>.
- [41] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. 2013. OAuth 2.0 Threat Model and Security Considerations. RFC 6819. (January 2013). doi: 10.17487/RFC6819. <https://rfc-editor.org/rfc/rfc6819.txt>.
- [42] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald, and Jörg Schwenk. 2014. Your Software at My Service: Security Analysis of SaaS Single Sign-on Solutions in the Cloud. In *th Edition of the ACM Workshop on Cloud Computing Security (CCSW '14)*. Association for Computing Machinery, New York, NY, USA, 93–104. ISBN: 978-1-4503-3239-2. doi: 10.1145/2664168.2664172.
- [43] Christian Mainka, Vladislav Mladenov, Tim Guenther, and Jörg Schwenk. 2015. Automatic Recognition, Processing and Attacking of Single Sign-On Protocols with Burp Suite. *Open Identity Summit*, 251, (October 2015), 117–131. ISSN: 16175468.
- [44] Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. 2016. Do Not Trust Me: Using Malicious IdPs for Analyzing and Attacking Single Sign-on. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 321–336. doi: 10.1109/EuroSP.2016.33.
- [45] Christian Mainka, Vladislav Mladenov, Tobias Wich, and Jörg Schwenk. 2017. SoK: Single Sign-On Security – An Evaluation of OpenID Connect. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 251–266. doi: 10.1109/EuroSP.2017.32.
- [46] MDN. 2021. Broadcast Channel API. Retrieved 10/28/2021 from [https://developer.mozilla.org/en-US/docs/Web/API/Broadcast\\_Channel\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Broadcast_Channel_API).
- [47] MDN. 2021. Channel Messaging API. Retrieved 10/28/2021 from [https://developer.mozilla.org/en-US/docs/Web/API/Channel\\_Messaging\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Channel_Messaging_API).
- [48] MDN. 2021. Creating and triggering events. (October 14, 2021). Retrieved 10/28/2021 from [https://developer.mozilla.org/en-US/docs/Web/Events/Creating\\_and\\_triggering\\_events](https://developer.mozilla.org/en-US/docs/Web/Events/Creating_and_triggering_events).
- [49] MDN. 2022. Proxy. MDN Web Docs. Retrieved 08/01/2022 from [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy).
- [50] MDN. 2020. Same-Origin Policy. MDN Web Docs. Retrieved 09/26/2020 from [https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin\\_policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy).
- [51] MDN. 2021. Window.postMessage(). Retrieved 10/28/2021 from <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>.
- [52] Gordon Meiser, Pierre Laperdrix, and Ben Stock. 2021. Careful Who You Trust: Studying the Pitfalls of Cross-Origin Communication. In *2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*. Association for Computing Machinery, Virtual Event, Hong Kong, 110–122. ISBN: 9781450382878. doi: 10.1145/3433210.3437510.
- [53] Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. 2016. On the Security of Modern Single Sign-On Protocols – Second-Order Vulnerabilities in OpenID Connect. (January 7, 2016).
- [54] Srivathsan G. Morkonda, Sonia Chiasson, and Paul C. van Oorschot. 2021. Empirical Analysis and Privacy Implications in OAuth-Based Single Sign-On Systems. In *20th Workshop on Privacy in the Electronic Society (WPES '21)*. Association for Computing Machinery, Virtual Event, Republic of Korea, 195–208. ISBN: 9781450385275. doi: 10.1145/3463676.3485600.
- [55] OWASP. 2021. Zed Attack Proxy (ZAP). Retrieved 12/02/2021 from <https://www.zaproxy.org/>.
- [56] Portswigger. 2021. DOM Invader. Retrieved 12/02/2021 from <https://portswigger.net/burp/documentation/desktop/tools/dom-invader/messages-view>.
- [57] Tamjid Al Rahat, Yu Feng, and Yuan Tian. 2019. OAUTHLINT: An Empirical Study on OAuth Bugs in Android Applications. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, (November 2019), 293–304. ISBN: 978-1-7281-2508-4. doi: 10.1109/ASE.2019.00036.
- [58] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. 2014. OpenID Connect Core 1.0 incorporating errata set 1. (November 8, 2014). Retrieved 10/27/2021 from [https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html).
- [59] Youssef Sammouda. 2020. Bad regex used in Facebook Javascript SDK leads to account takeovers in websites that included it. (December 31, 2020). Retrieved 08/29/2022 from <https://ysamm.com/?p=510>.
- [60] Giada Sciarretta, Roberto Carbone, Silvio Ranise, and Alessandro Armando. 2017. Anatomy of the Facebook Solution for Mobile Single Sign-on: Security Assessment and Improvements. *Computers & Security*, 71, (November 2017), 71–86. ISSN: 01674048. doi: 10.1016/j.cose.2017.04.011.
- [61] Ethan Shernan, Henry Carter, Dave Tian, Patrick Traynor, and Kevin Butler. 2015. More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations. In *Detection of Intrusions and Malware, and Vulnerability Assessment (Lecture Notes in Computer Science)*. Magnus Almgren, Vincenzo Gulisano, and Federico Maggi, editors. Volume 9148. Springer International Publishing, Cham, 239–260. ISBN: 978-3-319-20550-2. doi: 10.1007/978-3-319-20550-2\_13.
- [62] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau. 2019. MoSSOT: An Automated Blackbox Tester for Single Sign-On Vulnerabilities in Mobile Applications. In *ACM Asia Conference on Computer and Communications Security*. ACM, New York, NY, USA, (July 2019), 269–282. ISBN: 9781450367523. doi: 10.1145/3321705.3329801.
- [63] Soeul Son and Vitaly Shmatikov. 2013. The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24–27, 2013*. The Internet Society.
- [64] Marius Steffens and Ben Stock. 2020. PMForce: Systematically Analyzing postMessage Handlers at Scale. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. Association for Computing Machinery, New York, NY, USA, 493–505. ISBN: 978-1-4503-7089-9. doi: 10.1145/3372297.3417267.
- [65] Thomas Steiner. 2021. Trust Is Good, Observation Is Better: Intersection Observer V2. web.dev. (February 26, 2021). Retrieved 06/28/2021 from <https://web.dev/intersectionobserver-v2/>.
- [66] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. 2017. How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, (August 2017), 971–987. ISBN: 978-1-931971-40-9.
- [67] Avinash Sudhodanan, Soheil Khodayari, and Juan Caballero. 2020. Cross-Origin State Inference (COSI) Attacks: Leaking Web Site States through XS-Leaks. (January 31, 2020). arXiv: 1908.02204 [cs]. Retrieved 05/21/2021 from <http://arxiv.org/abs/1908.02204>.
- [68] San-Tsai Sun and Konstantin Beznosov. 2012. The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems. In *2012 ACM Conference on Computer and Communications Security (CCS '12)*. Association for Computing Machinery, Raleigh, North Carolina, USA, 378–390. ISBN: 9781450316514. doi: 10.1145/2382196.2382238.
- [69] Terjanq. 2022. Terjanq/same-origin-XSS: Same origin XSS challenge. Retrieved 08/16/2022 from <https://github.com/terjanq/same-origin-xss>.
- [70] Hui Wang, Yuanyuan Zhang, Juanru Li, and Dawu Gu. 2016. The Achilles Heel of OAuth: A Multi-Platform Study of OAuth-Based Authentication. In *32nd Annual Conference on Computer Security Applications (ACSAC '16)*. Association for Computing Machinery, Los Angeles, California, USA, 167–176. ISBN: 9781450347716. doi: 10.1145/2991079.2991105.
- [71] Rui Wang, Shuo Chen, and XiaoFeng Wang. 2012. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *33th IEEE Symposium on Security and Privacy (S&P 2012)*. IEEE, editor. doi: 10.1109/SP.2012.30.
- [72] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. 2013. Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, Washington, D.C., (August 2013), 399–314. ISBN: 978-1-931971-03-4.
- [73] Hanlin Wei, Behnaz Hassanshahi, Guangdong Bai, Padmanabhan Krishnan, and Kostyantyn Vorobyov. 2021. MoScan: A Model-Based Vulnerability Scanner for Web Single Sign-On Services. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, Virtual Denmark, (July 11, 2021), 678–681. ISBN: 978-1-4503-8459-9. doi: 10.1145/3460319.3469081.
- [74] Toru Yamaguchi, Nat Sakimura, and Nov Mataké. 2015. OAuth 2.0 Web Message Response Mode. Internet-Draft draft-sakimura-oauth-wrm-00. Internet Engineering Task Force, (October 18, 2015). 17 pages. <https://datatracker.ietf.org/doc/html/draft-sakimura-oauth-wrm-00>.
- [75] Ronghai Yang, Guanchen Li, Wing Cheong Lau, Kehuan Zhang, and Pili Hu. 2016. Model-based Security Testing: an Empirical Study on OAuth 2.0 Implementations. In *ACM on Asia Conference on Computer and Communications Security - ASIA CCS '16*. ACM Press, Xi'an, China, 651–662. ISBN: 978-1-4503-4233-9. doi: 10.1145/2897845.2897874.
- [76] Yuchen Zhou and David Evans. 2014. SSOscan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, (August 2014), 495–510. ISBN: 978-1-931971-15-7.
- [77] Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin. 2017. AUTHSCOPE: Towards Automatic Discovery of Vulnerable Authorizations in Online Services. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, Dallas Texas USA, (October 30, 2017), 799–813. ISBN: 978-1-4503-4946-8. doi: 10.1145/3133956.3134089.
- [78] Karsten Meyer zu Selhausen and Daniel Fett. 2022. OAuth 2.0 Authorization Server Issuer Identification. RFC 9207. (March 2022). doi: 10.17487/RFC9207. <https://www.rfc-editor.org/info/rfc9207>.
- [79] Ivan Zuzak, Marko Ivankovic, and Ivan Budiselic. 2011. A Classification Framework for Web Browser Cross-Context Communication. *arXiv:1108.4770 [cs]*, (August 2011). arXiv: 1108.4770 [cs].