RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Security and Privacy of Social Logins

Louis Christopher Jannett

Exposee – March 2, 2020.
Chair for Network and Data Security.

Supervisor: Prof. Dr. Jörg Schwenk
Advisor: Dr.-Ing. Christian Mainka
Advisor: Dr.-Ing. Vladislav Mladenov

hgi Lehrstuhl für Netz- und Datensicherheit

1 Introduction

OAuth is used to delegate the *authorization* of a Service Provider (SP) accessing end-users protected resources to an additional instance: the Identity Provider (IdP). Thus, end-users are no longer forced to share their login credentials with third-party SPs to grant access to all of their protected resources. [8, pp. 4-5]

OpenID Connect (OIDC) extends this concept by adding an identity layer on top of the OAuth protocol that additionally *authenticates* the end-user on the SP. This allows SPs to uniquely identify end-users and retrieve basic profile information. [15]

Various research about the formal specifications has been undertaken which lead to the introduction of new revisions and extensions of the standard. However, the real-world implementations of IdPs in the wild are more complex and vary from the standardized procedures. This results in an open attack surface on the different implementations of SPs and IdPs.

Since OAuth/OIDC IdPs can be considered as a single point of failure, simple implementation flaws most often have a dramatic impact on the SP's security, as shown by recent vulnerabilities [1]. Attacks may result in an attacker being able to log in as an arbitrary user on the SPs.

Thus, this thesis will examine the current implementations of Single Sign-On (SSO) in the wild by answering the following research questions:

1. How do real-world SSO protocols implemented by the most widely-adopted IdPs differ from the standard OAuth/OIDC specifications?
2. Are Single Page Applications (SPAs) using the postMessage Application Programming Interface (API) for cross-origin communication in the context of token exchange securely implemented?
3. What happens if `access_tokens` remain stored in the browser after user logout and what are the implications on the user's privacy?

1.1 Motivation

Although the OAuth/OIDC specifications clearly define protocol flows and security recommendations, real-world implementations often do not respect these guidelines.

Mainka et al. [10, p. 15] strengthened this observation in 2017, assuming that implementation flaws affecting the security will always exist. Developers are not aware of all potential security issues and thus will not follow the specifications perfectly.

Intercepting and analyzing real-world SSO login flows is challenging. Many non-standardized parameters & protocol flows, session management and complex browser communication via scripts must be considered. Thus, it is unavoidable to first analyze the real-world protocol flows by inspecting the relevance of parameters and messages. Finally, a profound security analysis can be performed.

As a motivation, the following practical observations stress the fact that SSO in the wild may vary significantly from the standard.

For instance, Facebook's SSO on Soundcloud[1] reveals a customized implementation of OIDC, nested in several requests with a wide range of non-standardized parameters (e.g. `skip_api_login`, `api_key`, `signed_next`, ...). Also, Facebook replaces the `id_token` with a `signed_request`. Since Facebook's login and consent page is opened in a new `window`, it uses the postMessage API to transfer the `access_token` and `signed_request` back to the `window` of Soundcloud.

Google offers multiple solutions for authorizing and authenticating users. For instance, *Google Sign-In* is advertised to be implemented with only a few lines of code [2] but exhibits deviations from the standard. In contrast, *Plain OAuth 2.0* tries to follow the specifications. [3]

Based on the concept of SPAs, Google's SSO flow is entirely performed from within the User-Agent (UA). For this type of public SPs, the standard requires the use of the OIDC Code Flow with Proof Key for Code Exchange (PKCE) [12]. However, Google skips the use of one-time redeemable `codes` protected with PKCE (`response_type = token id_token`). The `access_tokens`, signed `id_tokens`, and unsigned profile information are directly returned to the UA using postMessage API calls.

If needed, the UA is finally challenged to securely authenticate against a backend with the signed `id_token`. Google already warns developers to "[...] not accept plain user IDs [...] on your backend server. A modified client application can send arbitrary user IDs to your server to impersonate users, so you must instead use verifiable ID tokens to securely get the user IDs of signed-in users [...]". [4]

Outsourcing the responsibility of securely implementing the communication with backend servers to developers motivates further research. Thus, this thesis will first elaborate on how SSO is implemented and then inspect whether developers are securely implementing the custom guidelines of IdPs in SPAs.

As noticed above, the *postMessage* API [11] is heavily used in current SSO implementations in the wild to transfer tokens from IdPs to SPs. Recent research revealed many attack scenarios on this API (cf. Section 1.2). Although countermeasures have

[1] URL: `https://soundcloud.com`

been introduced, attacks are still possible if developers fail to implement it correctly. For instance, if a wildcard target origin is used in `service_provider.postMessage( id_token, "*")`, malicious third-party scripts can easily retrieve the tokens. Also, the responsibility for correctly checking the origin of a received message to prevent cross-origin attacks is delegated to the receiver functions. The latest attack on Facebook's SSO [1] used this API to send the stolen `access_token` to an attacker-controlled domain.

Finally, this thesis will explore the application of tracking mechanisms in the context of session management in SSO. In this scenario, `access_tokens` can be misused by SPs and IdPs by storing them permanently in the browser storage, even after the user logs out of the IdP. In the background, any SP the end-user visits could theoretically secretly redeem the stored `access_token` at the IdP. Thus, both parties are able to identify the end-user and the websites the end-user is visiting. This enables privacy-violating tracking mechanisms by abusing SSO session management and motivates extensive analysis.

1.2 Related Work

In 2012, Wang et al. [17] conducted the first field study on SSO systems by analyzing the traffic going through the browser. They recovered important semantic information to identify a total of 8 critical vulnerabilities in OIDC providers. Back then, the authors already criticized that the overall security of SSO systems in practice is alarming.

In 2016, Li and Mitchell [9] performed a large-scale practical study of Google's OIDC implementation. They examined a total of 103 SPs supporting Google's SSO. Several critical vulnerabilities allowing an attacker to log in as an arbitrary user on the SPs were discovered. The vulnerabilities are caused by a combination of Google's custom OIDC design, as well as design decisions made by developers.

Wang et al. [16] conducted a study in 2016 on how developers customize OAuth on different real-world platforms (e.g. Web, iOS, Android). They intercepted the traffic from an attacker's perspective to reconstruct the authentication mechanisms employed and identify potentially exploitable vulnerabilities. As a result, they found out that the applications lack sufficient verification mechanisms, resulting in multiple vulnerabilities.

The title of the paper "The Devil is in the (Implementation) Details [...]" by Sun and Beznosov [14] is as well its motivation. The authors analyzed the implementations of Facebook's, Microsoft's, and Google's IdPs and 96 SPs that supported Facebook's SSO. Several critical vulnerabilities that allow an attacker to impersonate a victim were disclosed. All were caused by the design decisions made by developers for implementation simplicity, disregarding the security.

Son and Shmatikov [13] performed a comprehensive analysis of postMessage origin checks in the wild. The authors analyzed postMessage receiver functions from the Alexa top 10,000 websites and identified 84 exploitable vulnerabilities, including Cross-Site-Scripting (XSS) and local storage injections. In the context of SPAs and SSO, this could lead to serious security issues since tokens are most commonly processes via JavaScript or stored in the browser's local storage.

In 2018, Guan et al. [5, 6, 7] developed the *DangerNeighbor* attack. It takes advantage of the fact that when a website contains multiple postMessage receiver functions, all functions can receive any message sent to this website. In practice, receivers are usually imported from third-parties via the `script` tag. Thus, they share the same origin as the hosting site. If an attacker can successfully inject a malicious receiver function, all messages sent to the website, including confidential `access_tokens`, can be eavesdropped. In a case study of this attack, a Chrome extension was developed that automatically injected a postMessage receiver function on the top 2,000 Alexa websites. The authors found out that 40% of websites using Facebook's OAuth and 23% of websites using Google's OAuth are vulnerable. They observed the `access_token` being leaked to the attacker-controlled receiver function. This thesis will examine if countermeasures are applied at all and if they are applied correctly.

2 Work Packages

2.1 Overview

The work packages are organized and categorized by the main three research questions of this thesis. First, an overview and protocol descriptions of real-world SSO protocols are worked out. Next, the security of SPAs in the wild regarding cross-origin communication in SSO is assessed. Afterward, the privacy of social logins is examined. Finally, the actual thesis is written.

2.2 Detailed Description

WP1 Overview & Protocol Description of Single Sign-On Protocols in the Wild

In this work package, the SSO implementations of IdPs in the wild will be examined and described. Therefore, differences (e.g. in flows, parameters) to the OAuth/OIDC standard and extensions will be highlighted.

Since the implementations of SPs will most likely not match the exact API documentation of the IdPs (e.g. more parameters, messages, custom scripts), manual traffic analysis with Burp will be conducted.

Also, different SPs of the same IdP may behave differently and implement their social logins in custom ways. This can be solved by first executing the SSO flows on multiple SPs for the same IdP. Then, similarities and differences are identified. Finally, the results are correlated and reduced to a uniform flow that holds for all SPs.

The tasks in this work package can be split up according to the different IdPs that are intended to be analyzed:

Task 1 Analyze Facebook's SSO protocol

Task 2 Analyze Google's SSO protocol

Task 3 Analyze Apple's SSO protocol

Optional: Additional IdPs, real-world SSO protocols in native Apps

Milestone 1: Finish description of real-world SSO protocols

WP2 Security of Single Page Applications in the Wild

In this work package, the session management in SPAs with SSO is analyzed in general. This includes the cross-origin communications for sign-in, as well as the mechanisms that are employed for tracking the current login status of the end-user. For instance, this thesis will check if the postMessage calls for cross-origin communication between the SP's and IdP' invisible `iframes` (e.g. `check_session_iframe`) with different security domains are securely implemented in SPAs.

The work package can be split up in the following tasks:

Task 1 Identify, understand & analyze session management in SPAs in the wild

Task 2 Identify, understand & analyze postMessage usage in SPAs

Task 3 Evaluate the security of session management and postMessage usage

Milestone 2: Finish analyzing the security of Single Page Applications concerning the usage of postMessage API calls in session management

WP3 Privacy in Single Sign-On Protocols

In this work package, the privacy implications resulting from incorrect/misused session management will be explored. The behavior expected to be found is that `access_tokens` or similar tokens are kept in the browser after the user logs out on the IdP. These tokens are then secretly and automatically redeemed by arbitrary SPs. This results in the SP's and IdP's ability to uniquely identify the user and track the accessed websites.

Task 1 Identify SPs with this behavior by using a list of public SPs

Task 2 Analyze traffic of websites supporting SSO with the target IdP on which a logout was performed

Task 3 Outline practical relevance and how tracking can be accomplished

Task 4 If results are positive, apply a large scale practical study of "Tracking with Social Logins". For instance, iterate over a list of SPs and check if stored `access_tokens` of logged-out sessions are transmitted.

Optional: Search for other methods in the wild to track users by abusing SSO mechanisms

Milestone 3: Finish analyzing the privacy implications of SSO in the wild

WP4 Writing the Thesis

Milestone 4: Finish the thesis

3 Organization of this Thesis

The thesis will be broadly divided into the following sections. Note that individual sections might be structured differently based on the outcome of the analysis (e.g. Security of postMessage). Also, the foundations might vary based on what is needed to understand sections 3-5.

1. Introduction
    - 1.1. Motivation
    - 1.2. Related Work
    - 1.3. Contribution
    - 1.4. Methodology
    - 1.5. Organization of this Thesis
2. Foundations
    - 2.1. Single Sign-On Basics
    - 2.2. OAuth 2.0
    - 2.3. OpenID Connect 1.0
    - 2.4. Session Management in OpenID Connect
    - 2.5. Single Sign-On in Single Page Applications
    - 2.6. postMessage API
3. Single Sign-On Protocols in the Wild
    - 3.1. Facebook Login
        - 3.1.1. Overview
        - 3.1.2. Protocol Description
    - 3.2. Google Sign-In
        - 3.2.1. Overview
        - 3.2.2. Protocol Description
    - 3.3. Sign in with Apple
        - 3.3.1. Overview
        - 3.3.2. Protocol Description
4. Security of Single Page Applications with Single Sign-On in the Wild
    - 4.1. Session Management in the Wild
    - 4.2. postMessage Usage in the Wild
    - 4.3. Security Evaluation
5. Privacy in Single Sign-On Protocols

5.1. Misuse of Single Sign-On for Tracking
5.2. Observations in the Wild
5.3. Privacy Implications

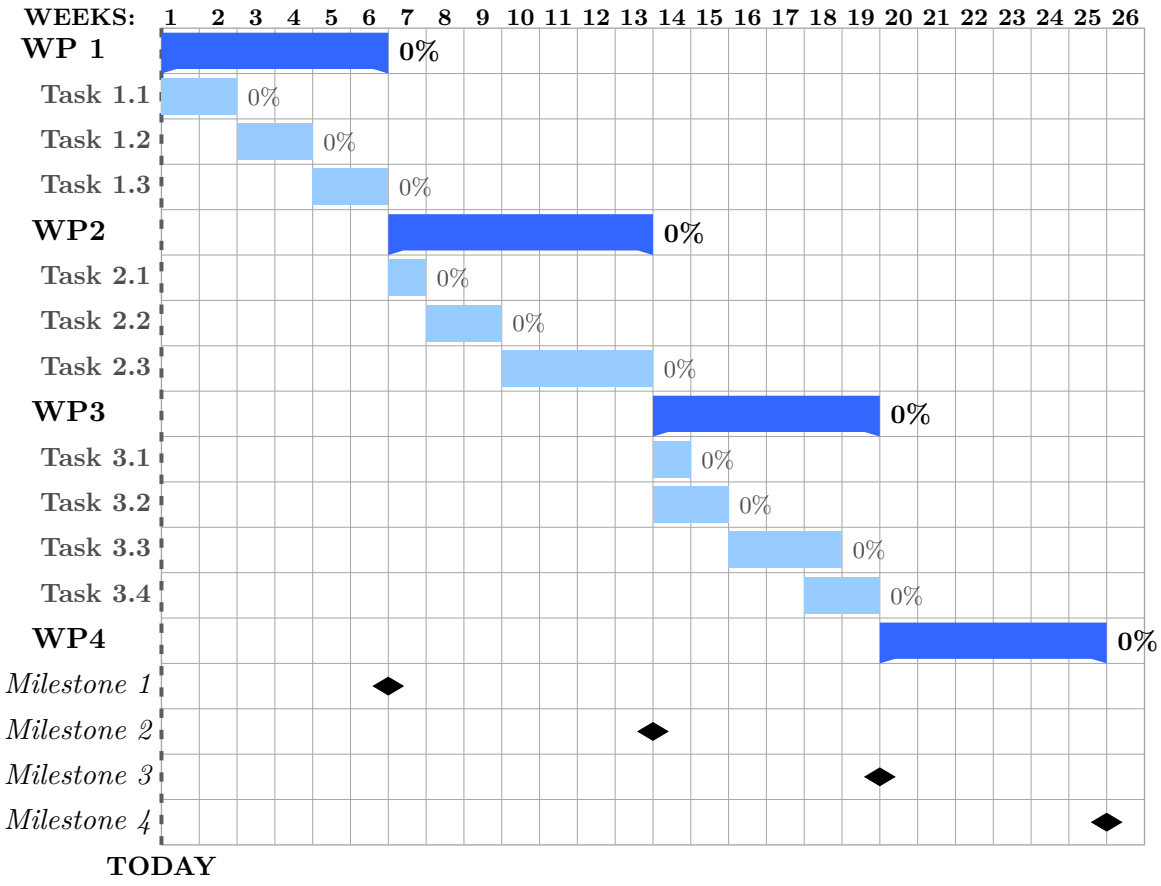
6. Conclusion and Future Work

6.1. Conclusion
6.2. Future Work

4 Timeline

In this chapter, a timeline depicting the starting and ending date for each work package is specified. This timeline includes the milestones as well.

4.1 GanttChart



Bibliography

[1] Amol Baikar. Facebook OAuth Framework Vulnerability, March 2020. URL `https://www.amolbaikar.com/facebook-oauth-framework-vulnerability/`.

[2] Google LLC. Add Google Sign-In to Your Web App. URL `https://developers.google.com/identity/sign-in/web`.

[3] Google LLC. Google Identity Platform, October 2017. URL `https://developers.google.com/identity/choose-auth`.

[4] Google LLC. Authenticate with a backend server, November 2019. URL `https://developers.google.com/identity/sign-in/web/backend-auth`.

[5] C. Guan, Y. Li, and K. Sun. Your Neighbors are Listening: Evaluating PostMessage Use in OAuth. In *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*, pages 210–211, August 2017. doi: 10.1109/PAC.2017.30.

[6] Chong Guan, Kun Sun, Zhan Wang, and WenTao Zhu. Privacy breach by exploiting postmessage in html5: Identification, evaluation, and countermeasure. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 629–640, 2016.

[7] Chong Guan, Kun Sun, Lingguang Lei, Pingjian Wang, Yuewu Wang, and Wei Chen. DangerNeighbor attack: Information leakage via postMessage mechanism in HTML5. In *Computers & Security*, volume 80, pages 291–305, 2018.

[8] Dick Hardt. *The OAuth 2.0 Authorization Framework*. Number 6749 in Request for Comments. RFC Editor, October 2012. doi: 10.17487/RFC6749. URL `https://rfc-editor.org/rfc/rfc6749.txt`. Published: RFC 6749.

[9] Wanpeng Li and Chris J Mitchell. Analysing the Security of Google's implementation of OpenID Connect. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 357–376. Springer, 2016.

[10] Christian Mainka, Vladislav Mladenov, Jörg Schwenk, and Tobias Wich. SoK: single sign-on security—an evaluation of openID connect. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 251–266. IEEE, 2017.

[11] Mozilla Corp. Window.postMessage() - Web APIs | MDN, September 2019. URL `https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage`.

[12] Nat Sakimura, John Bradley, and Naveen Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. Number 7636 in Request for Comments. RFC Editor, September 2015. doi: 10.17487/RFC7636. URL `https://rfc-editor.org/rfc/rfc7636.txt`. Published: RFC 7636.

[13] Sooel Son and Vitaly Shmatikov. The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites. In *NDSS*, 2013.

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

[15] The OpenID Foundation (OIDF). OpenID Connect Core 1.0, 2014. URL `http://openid.net/specs/openid-connect-core-1_0.html`.

[16] Hui Wang, Yuanyuan Zhang, Juanru Li, and Dawu Gu. The Achilles heel of OAuth: a multi-platform study of OAuth-based authentication. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 167–176, 2016.

[17] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *2012 IEEE Symposium on Security and Privacy*, pages 365–379. IEEE, 2012.