The Full Gamut of an Attack: An Empirical Analysis of OAuth CSRF in the Wild

Michele Benolli, Seyed Ali Mirheidari, Elham Arshad(✉), and Bruno Crispo

University of Trento, Trento, Italy
{seyedali.mirheidari,elham.arshad,bruno.crispo}@unitn.it

Abstract. OAuth 2.0 is a popular and industry-standard protocol. To date, different attack classes and relevant countermeasures have been proposed. However, despite the presence of guidelines and best practices, the current implementations are still vulnerable and error-prone. In this research, we focus on OAuth Cross-Site Request Forgery (OCSRF) as an overlooked attack scenario.

We studied one of the most recurrent types of OCSRF attacks by proposing several novel attack strategies based on different status of the victim browser. In order to validate them, we designed a repeatable methodology and conducted a large-scale analysis on 314 high-ranked sites to assess the prevalence of OCSRF vulnerabilities. Our automated crawler discovered about 36% of targeted sites are still vulnerable and detected about 20% more well-hidden vulnerable sites utilizing the novel attack strategies. Although our experiment revealed a significant increase in the number of OCSRF protection compared to the past scale analyses, over one-fourth are still vulnerable to at least one proposed attack strategy.

1 Introduction

OAuth 2.0 is an industry-standard protocol for authorization. It was released in 2012 as RFC 6749 and nowadays is pervasively used to manage authorization flows in web, desktop, mobile applications, and in smart devices. The protocol has been widely studied, and its theoretical and practical security has been covered extensively by the literature. OAuth was designed to enhance several aspects of the former client-server authorization model.

The OAuth 2.0 Threat Model and Security Considerations [26] and OAuth 2.0 Security Best Current Practice [16] documents are published to address the most common security issues and vulnerability scenarios discovered within concrete implementations of the protocol. However, despite the rich guidelines and the many mitigation proposed over time, several OAuth-based services are still subject to a wide range of security flaws. This because, those guidelines are not detailed enough to consider all possible settings that can lead to an attack, especially for what relates client-side parameters.

© Springer Nature Switzerland AG 2021
L. Bilge et al. (Eds.): DIMVA 2021, LNCS 12756, pp. 21–41, 2021.
https://doi.org/10.1007/978-3-030-80825-9_2

As reported by [23] CSRF vulnerabilities related to authentication and identity management services are extremely pervasive, even among the top-ranked domains. Our paper is mainly focused on a specific vulnerability, the CSRF attack against the `redirect_uri` [26], since it's one of the most popular concrete attack in OAuth implementations. The attack is well documented in the Threat Model document and it can lead to serious consequences, ranging from the disclosure of sensitive information to a malicious user [3] to the complete account takeover [10]. Our work extensively covers the details of this security threat, with a systematic analysis of its root causes and practical impact. We built an automated testing framework to evaluate the presence of the aforementioned vulnerability in a large number of popular sites that implement the Facebook login service. The rationale of our approach is to help developers to avoid implementation mistakes by providing the most comprehensive set of attack strategy such that developers are aware what implementation settings to avoid.

The outcome of our large-scale analysis is that more than a third of the tested sites were found vulnerable to at least one of the proposed attack strategy.

We selected only one attack because the purpose of the paper is not to find the highest number of vulnerabilities, but rather to demonstrate how to build a comprehensive set of attack strategies for an attack, considering scenarios and configurations that have been so far ignored or overlooked in the literature, This based on the wrong assumptions those scenarios were not significant. Our analysis proved they are indeed significant and contributed to find 20% additional vulnerabilities.

The paper makes the following contributions:

- To the best of our knowledge, we present the most comprehensive set of test cases to exploit OCSRF vulnerabilities, including novel attack strategies that stress all possible client-side status. They complement and integrate the guidelines provided by documents such as [16, 26] in helping OAuth developers to mitigate implementation mistakes.
- We designed a repeatable methodology and conducted an automated and large-scale analysis on 314 high-ranked sites to assess the prevalence of CSRF attack against the `redirect_uri` in OAuth implementations.
- The analysis discovered that about 36% of targeted sites are still vulnerable and detected about 20% more well-hidden vulnerable sites utilizing the novel attack strategies.

2 Background

This work primarily focuses on a specific OAuth vulnerability, that can lead to a cross-site request forgery attack. For a thorough understanding of the risks and consequences related to this vulnerability, this section provides a brief background on OAuth and CSRF attacks in the context of OAuth. Threat model and its impact are described as well.

2.1 OAuth

OAuth is an authorization protocol and does not handle user authentication. However, authentication protocols can be built on top of it [19]. Many identity providers (IdP) such as Google and Facebook use OAuth to allow their users to share identity and personal information with third-party websites and applications (clients).

The OAuth 2.0 specification describes different methods for a client application to obtain an access token and consequently the access to user's protected resources. The four grant types are authorization code, implicit, resource owner password credentials, and client credentials. Each grant type is optimized for a particular use case. In this research, we are only concerned with the authorization code and implicit grant flows.

2.2 Login CSRF

In a login cross-site request forgery, the attacker deceives the victim into executing a cross-site request to the login endpoint of a target website. The attacker uses its own credentials to forge the login request. If the attack succeeds, the server issues a session cookie for the browser of the victim. As a result, the victim is logged into the target website with the account of the attacker [3]. There has been several studies [13,21,23,30] analysing the login CSRF attacks.

At first sight, the attack may appear quite innocuous. Generally, a cross-site request forgery attack concerns operations performed on the victim's protected resources. In the login CSRF, the attacker exploits an application flaw to deceive the users into performing some unintended operations inside the attacker's account. The browser state is changed after the execution of the attack, and the victims may be completely unaware of the fact they are using an account owned by someone else. As a result, they may upload sensitive documents, share credit card numbers and other personal data with a malicious user.

2.3 State Parameter

According to OAuth 2.0 specification [9], the client must implement CSRF protection for its redirection URI. Any request sent to the redirection endpoint must include a value that binds to the user-agent's authenticated state. This value `state` parameter which can be performed in OAuth 2.0 flow. The state parameter, to prevent CSRF attacks, should be a non-guessable randomly generated sequence of characters. However, the presence of the parameter does not guarantee the security of the client against a CSRF attack. A wrong validation or a mishandling of the parameter may lead to the vulnerability of the application, as evidenced in [30].

3 OAuth Cross Site Request Forgery

In the context of OAuth 2.0, a successful cross-site request forgery can allow an attacker to obtain authorization to resources protected by the protocol, without

the consent of the user. A recurrent type of login OCSRF is the OCSRF attack against `redirect_uri` [26], where the victim is logged into an account controlled by the attacker. As a direct consequence, all the operations performed by the victim are unconsciously accomplished inside the attacker's session and the result of these actions can potentially be disclosed to the attacker.

3.1 Threat Model

Figure 1 represents the main steps of the OCSRF attack against `redirect_uri` considered in our large-scale analysis. The attack starts with the victim's browser opening a malicious web page (1). At the loading page, the crafted request is generally launched by the browser automatically (2). The OAuth flow, initiated by the attacker, is then completed on the victim's side. The identity provider exchanges the received code for an access token and returns it to the client (4). At this point, the client can use the token to access the information needed to authenticate the user. Since the flow is initiated by the attacker, the login is performed using the attacker's account.

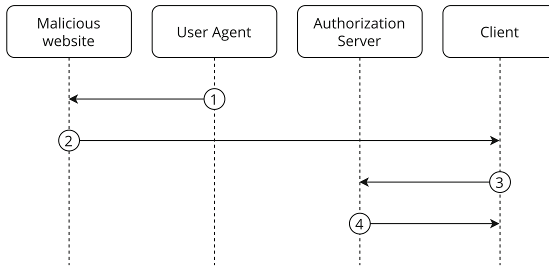

Fig. 1. Main steps involved in the OCSRF attack against `redirect_uri`

3.2 Impact

Some sites allow their users to register several OAuth provider logins, linked to the principal account. It represents an alternative and simpler way to access the application. In this scenario, if one of the implemented OAuth flows is insecure, a login OCSRF attack may lead to an account takeover. The account linking feature can be exploited to gain full access to the victim's data. The attack flow was first discussed by Egor Homakov [10]. The attack is possible only if certain preconditions are satisfied. The attack can only be executed against a registered user on the target site. At the end of the attack, the account owned by the attacker is linked to the victim's account. As a result, the attacker can access the victim's account on the client with the identity provider's profile used in the attack.

3.3 Enabling Factors

Several factors can influence the success rate of the OCSRF attack against `redirect_uri`. What happens if the victim is registered to the vulnerable application? Is the attack feasible even if the user never visited the domain before? If the victim is already authenticated to the website, is the attack prevented? Having these questions answered is important to better understand the impact of OCSRF in real-life scenarios. To be exploitable, the OCSRF attack against `redirect_uri` does not require the victim to be authenticated on the target application. Frequently the attack works even if the victim never visited the site before. However, the presence of cookies, previously set by the target site in the browser, can alter the outcome of the attack. In our analysis, we investigated this hypothesis running all the test scenarios with three different victim browser status as follows: a) No cookie, b) Visitor (unauthorized) cookies and c) Authorized cookies. We designed different attack strategies utilizing above-mentioned victim browser status which would be discussed in detail in Sect. 5.3. In the rest of the paper, for brevity, OCSRF attack refers to the OCSRF attack against `redirect_uri`.

4 Related Work

The security of OAuth 2.0 has been widely examined in the literature. Several theoretical studies (e.g. [2,7,20,29]) use abstract models to evaluate the security of the OAuth protocol. A downside of theoretical approach is that it does not allow to discover the vulnerabilities resulting from implementation errors.

Many empirical works have been done on the security of OAuth-SSO (e.g., [1,5,6,18,25,28,31]) either by developing web-based tools or evaluating the risk in real-world implementations.

A similar approach is employed by Li and Mitchell [13] to analyse the security of SSO implementations based on OAuth 2.0. Regarding the CSRF attack against the `redirect_uri`, authors found a significant fraction of clients are not implementing any countermeasure. The detected security issues were manually inspected and led to the generation of several case studies.

Sumongkayothin et al. present OVERSCAN [24], a security scanner able to identify missing parameters within the OAuth 2.0 protocol, analysing the traffic between the browser and the web application. Part of this analysis required manual inspection. The main limitation of the manual approach is scalability. The lack of automation makes the inspection process extremely time consuming and the limited size of the resulting sample makes it difficult to generalize the findings and distill error patterns.

Calzavara et al. [4] designed and implemented a browser-side security monitor for web protocols, called WPSE, to prevent nine attacks violating the security properties of OAuth. However, WPSE cannot prevent certain classes of attacks, including automatic login CSRF attacks, network attacks which are not observable by the browser and impersonation attacks.

Yang et al. [30] propose a model-based approach for the automated discovery of vulnerabilities in OAuth 2.0 implementations, called OAuthTester. To overcome the limitations of previous theoretical approaches, OAuthTester starts building a state machine from the protocol specifications, but then enhance the state machine and fills the gaps due to the ambiguities of the specification by observing traffic traces of the OAuth flow and the server state. However, as said by the authors, they can observe only traffic over HTTP, so they cannot gain the all knowledge we used in our approach to design the attack strategies. As a results they do not detect vulnerabilities that we detect with our analysis.

Shernan et al. [21] perform a large-scale analysis to assess the presence of CSRF vulnerabilities in real-world deployments of OAuth. The analysis on the Alexa Top 10K sites reveals that 25% of sites using OAuth were vulnerable to CSRF attacks. A significant limitation of this approach is represented by the metric used to assess the occurrence of CSRF vulnerabilities. A lot of sites were excluded from the analysis simply because of the existence of the `state` parameter in the authorization URL. As we show in this paper, the mere presence of the `state` value does not guarantee protection against CSRF attacks.

Sudhodanan et al. [23] present a comprehensive study on the different types of authentication CSRF reported in the literature. For identification of strategies in order to detect and reproduce each vulnerability, they used the same browser to simulate the interaction between the attacker and the victim, which led to missing some additional scenarios regarding to victim's browser states at the time of the attack. Our approach consider additional attack strategies. Instead of using the same browser, we totally separated the environment in which attacker and victim operate. In place of performing the attack only in a clean browser session, we also performed tests in presence of visitor and authorized cookies; which is not considered in their analysis.

5 Methodology

We designed a repeatable methodology to discover and validate OCSRF vulnerabilities in targeted sites. As depicted in Fig. 2, our methodology has three phases: 1. target selection, 2. measurement setup 3. OCSRF detection. We developed a tool based on Python-Selenium to automatically select targets and test different OCSRF scenarios.

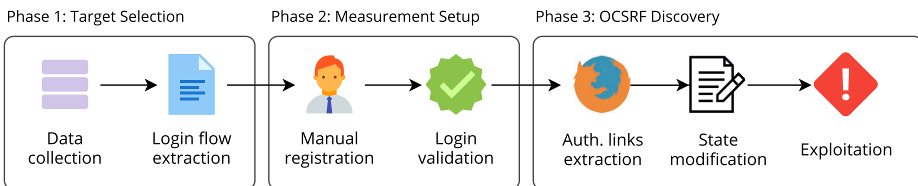

Fig. 2. Abstract view of OCSRF detection methodology.

5.1 Phase 1: Target Selection

Step 1: OAuth Login Detection. For extracting the initial seed set of candidate sites using OAuth login, we develop a browser-based crawler to visit sites in the initial seed set (e.g., Alexa Top 50K) in April 2020. The crawler is designed in a way that extracts initial OAuth login links for specific popular providers via checking the presence of OAuth standard parameters in all extracted links.: `response_type`, `client_id`, and `oauth`. The string «`oauth`» is commonly contained in the URL of authorization endpoints and its presence is a good indicator of the existence of an OAuth-based process. All these parameters are used by the crawler in the detection phase, to classify the links and identify the different login systems built on top of the OAuth protocol. Since many sites use JavaScript which requires interaction with users to trigger OAuth login, we develop a browser-based crawler to increase the detection rate.

Step 2: OAuth Flow Extraction. In order to remove false positives and extract OAuth redirection flows properly, the crawler follows all extracted and selected links. If the crawler lands on any well-known identity provider we will add the site to our candidate list, which would be later used to test our OCSRF attack strategies. A keyword-based approach is used to detect the Login/Sign-in buttons (these elements usually contain some known keywords to identify the login action and the identity provider). Extracted flows would later fed into next phases.

5.2 Phase 2: Measurement Setup

Step 1: Manual Registration. We follow the extracted OAuth links and create two sets of test accounts (victim and attacker) for each targeted site. Since the information provided by the external identity provider is not sufficient for the account creation process in many targeted sites, manual data entry is necessary. We adopt previously proposed technique [17] to populate attacker and victim accounts with unique information (e.g., name, email, user identifier, phone number, profile logo, etc.) and use them in next steps as `markers`.

Step 2: Login Validation. To verify the login steps, the crawler uses the login information gathered in the first phase to initiate the OAuth login trail. It reaches the authentication page and enters the credential automatically. At this point the flow is complete, and the browser is redirected to the target site's landing page.

OCSRF attack detection requires a victim to login as an attacker to the targeted site. The detection crawler should be capable of detecting the forged login to the attacker's account. In this regard, a learning process is developed for the crawler to automatically complete and learn the login processes for both attacker and user accounts. In the learning process, the crawler scans the HTML code of the landing page and looks for specific user-related strings. We presume the presence of some predefined unique `markers`, visible only as a result of a valid login to each account (which is populated to each account in registration step).

5.3 Phase 3: OCSRF Discovery

The main goal of this phase is to discover exploitable sites. The crawler is designed in a way to discover various implementation flaws in `state` validation (described in the step 2). In the first step, the crawler follows the OAuth flow, logs into the attacker account and extracts the authorization response links. In the second step, the crawler applies different modifications based on five attack strategies on the extracted authorization link. In the last step, the different victim browser status is exploited with modified links.

Step 1: Authorization link Extraction. Since the successful exploitation of OCSRF needs an attacker authorization response link including authorization code, `state` etc., the crawler initially follows OAuth login and obtains an attacker authorization response from the identity provider. We develop a browser extension to allow the crawler to record the attacker authorization link from the identity provider and halt the OAuth flow immediately. In other words, the generated authorization link is recorded and the OAuth flow is stopped before redirection to the target site. The extracted link will be modified in the next steps to discover vulnerable sites.

Step 2: `state` Modification. The extracted authorization link would be modified via going through five attack strategies. All attack strategies are performed mainly based on modifications on `state`, as a result of which attack URLs would be created. The first scenario is applied to the subset of sites in which a `state` is not present in the authorization link. In other scenarios, attack strategy would build further attack URLs by manipulating the `state` value as enumerated as follows.

0. **No `state`.** The link is sent unaltered to the victim if the original link does not contain a `state`.
1. **Empty `state`.** The `state` value is replaced with an empty string.
2. **Lack of `state` validation.** The value of the `state` is replaced with a randomly generated string.
3. **Unlinked `state`.** The link including `state` is sent unaltered to the victim.
4. **Missing `state`.** The `state` is removed.

In the first attack strategy, the authorization response link obtained at the first stage remains unchanged. In order to build other test cases, the testing strategy would manipulate the value of `state` value by either replacing it with an empty string, substitute it with a randomly generated string or keep the same value. Last attack strategy would completely remove the `state` parameter. In both strategies 0 and 3 the attacker would deliver the attack link unchanged to the victim. The strategies 1 and 2 rely on different alterations of the `state` value. In strategy 1, the content of the parameter is replaced with an empty string while attack strategy 2 replaces the value with random string. Finally, in the last strategies the `state` value and parameter name is completely removed.

Step 3: Exploitation. Each of the attack URLs generated in the previous step would be opened in a separate browser. We propose several OCSRF test cases

based on above strategies to determine whether a site is exploitable or not. In this regard, above strategies assess various victim browser status. Each of them is performed on three different victim browser status:

(a) **Status A. No Cookie,** when the victim opens the attack URL, there is no cookie related to the targeted site in the victim browser. In other words, either victim never visited the targeted site in the past or uses a new/history-cleared browser. Obviously, no cookies will be sent to the server when attack URLs are requested.
(b) **Status B. Visitor Cookies,** If the victim visited the targeted site in the past and visitor or unauthorized cookies have been set in the browser, the victim browser adds them to all requests. In this case, the crawler visits the first page of candidate site and stores all cookies before requesting the attack URLs.
(c) **Status C. Authorized Cookies,** If the victim is already authenticated to the targeted site, authorized cookies have been set on the victim browser. To simulate this test case, the victim has been authenticated by our crawler through logging in the victim's account, before requesting the attack URLs.

Each of the created attack URLs, obtained from applying previously-mentioned strategies, would be tested on each and every browser status defined above. As we have five different attack strategies and three possible victim browser status, we would end up with 15 test cases which would be exploited for each site. We later open all attack URLs inside victim browsers. We consider a test case to be successful if the attacker's marker is observed inside the victim's browser.

In a nutshell, each test case could be considered as the following three-step process.

1. Extract an attacker valid authorization response.
2. `state` parameter modification based on attack strategies.
3. Simulation of OCSRF attack on one of victim browser's status

5.4 Ethical Consideration

All test cases were performed with accounts specifically generated for this purpose. We never tried to exploit user accounts outside of our control. In all vulnerability assessment phases, our crawler never injected, sent or stored any malicious payload to candidate sites. In order to evade detection by bots detector [27], less than 100 pages of each candidate site was visited slowly in the data collection phase. We also developed Selenium crawler to complete the authentication steps and simulate a real user browser session. The number of requests involved in all test cases are significantly low, and the examined websites did not suffer from excessive bandwidth consumption. Moreover, all tests were conducted on the entire set of candidate sites therefore none of them has repeatedly been scanned in a short period of time.

Responsible Disclosure. Since the impact of the discovered vulnerabilities are severe, we reported the site owners using recommended notification techniques [12,22]. Additionally, we tried to disclose the vulnerabilities to those sites for which a centralized reporting system such as Hackerone [8] can be used, as these promise an increased success rate over attempting direct notification.

6 Analysis

In this section, we present the results of the empirical analysis and discuss them in detail. We also independently evaluate the results of each attack strategy and test case. This approach gives us the opportunity to properly focus on individual case studies among exploitable sites. The section would be concluded with the presentation of noteworthy observations.

6.1 Measurement Overview

Dataset. We fed our crawler with the Alexa Top 50K sites and analyzed the first page of them to extract the list of candidate sites with OAuth login. Since most of the discovered sites support different identity providers in their authentication pages, we only targeted one of their implementations and selected the most popular one, Facebook. The crawler discovered 539 sites with Facebook login. In the next step, we tried to create two sets of accounts (victim and attacker) and recorded the successful OAuth flow on each site. We narrowed down the dataset to 314 due to exclusion of sites with incomplete account registration (e.g., Social Security Number, credit card, etc.) and unsuccessful account verification.

Alexa Ranking. Our crawler analyzed all fifteen proposed test cases on target dataset and discovered 114 out of 314 (%36.3) sites to be exploitable by at least one test case. Given the distribution of the targeted and vulnerable sites across the Alexa Top 50K, it is noteworthy that about 32% of the sites among the Top Alexa 1K are vulnerable. Sites with higher Alexa ranking are slightly more vulnerable, but no specific major correlation among different buckets has been observed.

Categories Based on Presence of `state`. The candidate sites have been categorized based on absence or presence of `state` parameter within the recorded authorization request. For the former category, as mentioned in Sect. 5.3, our crawler directly exploit site without `state` and no modification applied on the attack URLs. However, on the latter category, due to presence of state parameter, 15 different attack scenarios have been tested. We will discuss the result of both categories in Sect. 6.2 in detail.

1. The first category, 44 out of 314 (14.0%) sites, do not use `state`, which shows a significant increase in utilization of `state` compare to past large scale analyses [3,11,14].

2. The second category, 270 out of 314 (85.9%) sites, are using `state`. Although, this indicates a significant increase in OCSRF protection compared to the past studies [3,11,14,15], our crawler detected 73 out of 270 (27.0%) exploitable sites utilizing different test cases.

Table 1. Number of exploitable sites in Facebook by OCSRF for each attack strategies

#	No cookies (a)	Visitor cookies (b)	Auth. cookies (c)	All
0	33 (10.5%)	41 (13.1%)	23 (7.3%)	41 (13.1%)
1	34 (10.8%)	33 (10.5%)	23 (7.3%)	41 (13.1%)
2	30 (9.6%)	40 (12.7%)	23 (7.3%)	40 (12.7%)
3	49 (15.6%)	63 (20.1%)	36 (11.5%)	64 (20.4%)
4	33 (10.5%)	34 (10.8%)	24 (7.6%)	40 (12.7%)
Total	91 (29.0%)	105 (33.4%)	62 (19.7%)	114 (36.3%)

Attack Strategies. Table 1 shows the number of exploitable sites to each attack strategy. As shown, the «attack strategy 3: Unlinked `state`» has the highest success rate (20.4%) in all victim browser status. In this attack strategy, as previously described in Sect. 5.3, the victim visited a crafted attack URL with an attacker's valid and unused `state`. It means lack of proper relation between the victim browser and generated `state` is the most common implementation mistake. Interestingly «attack strategy 1: Empty state» has the second rank which means some sites mistakenly accept the authorization link with a null `state` value.

Test Cases. Since visitor cookie is the most vulnerable status which makes highest success rate (20.4%) and «attack strategy 3: Unlinked `state`» is the most effective attack strategy, test case «3b» has the highest detection rate. Our crawler detected 63 out of 270 (20.1%) sites to be exploitable with it. Test cases «1c» and «2c» had the lowest detection rates, most probably because targeted sites do not accept new OAuth login when user is authenticated.

Table 2. Classification of exploitable sites in Facebook by OCSRF - The first category of candidates (with Absence of `state` parameter)

#	0a	0b	0c	Sites
1	●	●	●	17 (38.6%)
2	●	●	○	16 (36.4%)
3	○	●	●	6 (13.6%)
4	○	○	○	3 (6.8%)
5	○	●	○	2 (4.5%)
Total	33	41	23	44

Victim Browser Status. We tested each attack strategy with three different victim browser status. Our crawler detects unique exploitable cases in each browser status. Previous researches only test the OCSRF in a clean browser without presence of any cookie [23] or only with visitor cookie [30]. In this research, our crawler was able to detect 23 out of 114 (20.2%) more exploitable OCSRF cases compared to test case «a: No cookies» through utilizing different browser status and 9 out of 114 (7.9%) compared to test case «b: Visitor cookies». Applying all of the browser status together with attack strategies have been done for the first time to the best of our knowledge.

Based on our results presented in Table 1, the presence of visitor cookie in victim browser increases the chance of finding exploitable cases significantly. Even though it is common that sites with authorization cookies are less vulnerable, we observed test cases that unexpectedly were vulnerable only in this specific test cases, which would be discussed in Sect. 6.2.

Table 3. Classification of exploitable sites in Facebook by OCSRF - The second category of candidates (with Presence of `state` parameter)

#	1a	1b	1c	2a	2b	2c	3a	3b	3c	4a	4b	4c	Sites	Sites/Const state
1	○	○	○	○	○	○	○	○	○	○	○	○	197 (73.0%)	1 (5.9%)
2	●	●	●	●	●	●	●	●	●	●	●	●	18 (6.7%)	4 (23.5%)
3	○	○	○	○	○	○	●	●	●	○	○	○	11 (4.1%)	5 (29.4%)
4	●	●	○	●	●	○	●	●	○	●	●	○	7 (2.6%)	0
5	○	○	○	○	○	○	●	●	○	○	○	○	6 (2.2%)	1 (5.9%)
6	○	●	○	○	●	○	○	●	○	○	●	○	5 (1.9%)	1 (5.9%)
7	○	○	○	○	○	○	○	●	○	○	○	○	4 (1.5%)	3 (17.6%)
8	●	○	○	○	○	○	○	○	○	●	○	○	4 (1.5%)	0
9	○	○	○	●	●	●	●	●	●	○	○	○	3 (1.1%)	0
10	○	○	○	○	●	○	○	●	○	○	○	○	3 (1.1%)	0
11	●	○	○	○	○	○	○	○	○	○	○	○	2 (0.7%)	0
12	●	○	●	○	○	○	○	○	○	●	○	●	2 (0.7%)	0
13	○	○	○	●	●	○	●	●	○	○	○	○	2 (0.7%)	0
14	○	●	●	○	●	●	○	●	●	○	●	●	2 (0.7%)	1 (5.9%)
15	○	○	○	○	○	○	●	●	●	●	●	●	1 (0.4%)	1 (5.9%)
16	○	○	○	○	○	○	●	○	○	○	○	○	1 (0.4%)	0
17	●	●	●	○	○	○	○	○	○	●	●	●	1 (0.4%)	0
18	○	○	○	○	○	○	○	●	●	○	○	○	1 (0.4%)	0
Total	34	33	23	30	40	23	49	63	36	33	34	24	270	17

6.2 `state` Parameter

As mentioned, there are two categories of candidates based on the presence of `state` parameter within the recorded authorization request, which would be analysed and explained separately in this section.

Absence of `state`. Interestingly, 44 out of 314 (14.0%) of sites do not set `state` and our crawler detected 33 (75.0%), 41 (93.1%) and 23 (52.2%) sites are vulnerable to test cases «0a», «0b» and «0c» respectively. In some cases, the absence of visitor cookies led to errors in the OAuth login flow, and this contributes to explain the lower number of vulnerabilities found in status «a» than «b».

Interestingly, all exploitable sites are also exploitable to «b» while about half of them are not exploitable when there is an authorized cookie. The classification of exploitable sites are listed in Table 2. Each row represents one pattern w.r.t different test cases (1a, 1b, etc.). A filled circle in each entry indicates successful exploitation. The `Sites` column shows the total number of sites which have been found exploitable via the indicated pattern in corresponding row. For example, 3 out of 44 sites were not exploitable to any of test cases, so on. While only 17 sites are vulnerable to all three test cases, there are two sites that are only exploitable when the visitor cookies are present. It means successful exploitation of them requires the victim browser to add only unauthorized cookies in the Attack URL.

In contrast to other researches, absence of `state` does not guarantee success exploitation of OCSRF, as other enabling factors can prevent targets from being exploited. In order to remove the false positives, our crawler analyzed all 44 sites in the first category of candidates. Unexpectedly, 3 sites were not exploitable. Two out of three sites use encoded and nonstandard parameters in the `redirect_uri` and implement proper validation to check if the OAuth flow initiated with the same browser. At the time of writing this paper, Facebook doesn't allow developers to set arbitrary parameters to `redirect_uri` as the full redirect URL should be reserved and the OAuth flow is blocked if there is any change in `redirect_uri`. It seems Facebook still allows old implementations to use nonstandard parameters in `redirect_uri`, probably for backward compatibility reasons. Anyhow, further investigation into the exceptions of the Facebook OAuth implementation is beyond the scope of this research.

The third secure expects the flow to be completed in a popup window, which is not opened by the crawler during the attack execution. The JavaScript code running on the client-side fails due to the absence of an opener parent window and the attack is consequently blocked in the browser. We consider this site as a secure one despite the absence of adequate protection against OCSRF. We will discuss related case studies in Sect. 6.4.

Presence of `state` Attack. Presence of the `state` does not mitigate OCSRF vulnerabilities. We summarized each exploitable pattern which was observed during our experiment on 270 sites in our candidate set in Table 3. About 73% of sites are not exploitable to any of the proposed test cases. In many sites, this is due to a correct implementation of the OAuth flow. Some secure instances notify the user about the OCSRF attack, others simply display a generic authorization error or do not perform any action. It should be noted that the group of 197 site marked as not exploitable by the crawler may contain a small fraction of false negatives. This hypothesis is supported by some evidence presented later

in the analysis. For this reason, the number of vulnerabilities identified in our tests must be considered as a lower bound. Details would be discussed in the following section.

6.3 Case Studies

In this section, different test cases used during this research would be explained along with notable case studies of each attack strategy. It is worth mentioning that in this section the second category, presence of `state` parameter, is studied.

Empty and Missing `state`. In attack strategy 1, the value of the `state` in the authorization response is replaced with an empty string. At the beginning of the flow, the site generates a valid `state` to identify the authorization request. If the authorization response contains an empty `state` value, the application is supposed to not accept it and block the OAuth flow. The same approach applies to attack strategy 4, in which the `state` parameter – not only its value – is entirely removed from the authorization response URL. One of the manually analyzed sites has been discovered to be exploitable only to attacks 1 and 4, as illustrated in Table 3, classification 17. This result shows that when the parameter is present, `state` is handled supposedly and would be verified by the application. However, when the `state` value is empty or the parameter is missing, the validation would be bypassed and the flow successfully be accepted. The application source code is not directly available. However, we can get an insight into the internal logic of the `state` validation algorithm by analyzing the site reactions in response to different inputs.

A couple of sites are exploitable only via attack strategy 1 but not the 4th (Refer to Table 3, classification 11). The validation process checks the presence of a parameter called `"state"` in the authorization response and blocks the flow if it is not found. However, an empty `state` is accepted as valid and leads to the flow completion. The reverse is still possible when a site is exploitable with attack 4 and not to 1 (Refer to Table 3, classification 15). As an instance, we found a case study in which the verification succeeds only in presence of a valid `state`, while it could be bypassed if the parameter was not provided. The empty `state` supplied in the first test scenario was considered invalid by the application and caused the flow to be halted. Furthermore, we also found 6 exploitable sites in which the only performed validation is related to the presence of the `state` parameter inside the authorization response (Table 3, classification 9 and 10). The client application does not accept requests with a missing or empty `state` parameter, but even a random value is enough to bypass the validation.

The difference between attack strategy 1 and 4 is subtle and the results are almost overlapping. But the insight provided by above-mentioned unexpected results would be to take both attack strategies into account to discover related vulnerabilities to a great extent.

Lack of `state` Validation. In attack 3, the authorization response received by the attacker is maintained unchanged and sent to the victim. The test is performed to assess the absence of a valid relation between the `state` and the

user's session. If the `state` is not handled properly during the generation of the authorization request, the application does not have enough information to perform correct validation in the subsequent steps of the flow. The site is not able to understand whether the authorization response was issued by the identity provider for the current user or if someone else initiated the request. As a result, the client may accept all the `state` values produced by the application as valid.

As illustrated in Table 1, attack strategy 3 is the most successful one. More than 20% of the candidate site are vulnerable to scenario «3a», «3b», or «3c». This can be justified by the inherent complexity of implementing a valid relation between the browser session and the `state`, which requires to generate and store a random token and proper management of that in the validation phase. Even though the RFC clearly describes the role and operation of the `state` parameter, the documentation provided by different identity providers are not often sufficiently precise and detailed. 23 out of 114 (20.2%) exploitable sites are only vulnerable to attack strategy 3 (Table 3, classifications 3, 5, 7, 16, and 18). For these applications, arbitrary `state` values are correctly rejected by the validation method, but valid states with incorrect associated to user sessions are erroneously not refused. Eleven sites are vulnerable to all configurations of attack strategy 3 (Table 3, classifications 3).

Unlinked `state`. In attack strategy 2, the `state` parameter produced by the client application is replaced with another string which is a random permutation of the initially generated value. The new parameter has the same length as the original and the same character set. The purpose of this strategy is to understand if using an invalid `state` is sufficient to bypass the OCSRF protections implemented by the examined sites. Our crawler detects total number of 30 out of 114 (26.3%), 40 out of 114 (35.1%) and 23 out of 114 (20.2%) exploitable sites to be vulnerable to test cases «2a»,«2b», and «2c». Presence of visitor cookies increases the attack success rate similar to other attack strategies.

It can be easily noticed from Table 3 that the results of attack strategies 2 and 3 are strongly related. There are no sites discovered to be vulnerable only to 2, and the sites vulnerable to this attack constitute a proper subset of the ones vulnerable to the third scenario. Although it does not add any item to the set of vulnerable domains, the second scenario gives remarkable indications about the nature of the validation performed. For instance, looking at the attack results reveals the possibility of a completely incorrect validation from a session association issue. A particular case in attack 2 is represented by sites that use a `state` consisting of a single character, or a sequence of N identical characters. In this scenario, the generation of a different permutation of the original string is not feasible and the attack cannot be performed as originally described. Among the samples considered, there are two sites with `state` of length one. The characters used are underscore «_» and slash «/», respectively. The site using «/» was found vulnerable to attack 3b. The attack succeeded even if «/» is substituted with a different arbitrary character. In the other site, the replacement of «_» with a random character prevented the attack from being executed successfully.

Based on above-mentioned implementation mistakes, we recommend to use a `state` value which is not guessable and is randomly produced. Moreover, it

is required `state` to be in correct association with the user session in order to avoid OCSRF vulnerability.

6.4 Notable Observations

Constant `state`. The OAuth 2.0 specification clearly states that the `state` parameter must be one time use and a random string. This requirement is necessary to protect applications from brute-force attack. Some sites do not follow these instructions and include a fixed and constant `state` in the OAuth authorization request which would not change for different users and browser sessions. These websites are not able to distinguish between a legitimate authorization response created for the victim and a response forged by the attacker. We visited all candidate sites twice from two different browser sessions and compared the `state` values in order to identify this implementation problem. If the `state` remains unchanged, the site is potentially vulnerable to a "state reuse" attack. Our crawler collected and stored all authorization requests. We later extracted the `state` values from the URL and compared them to each other. The analysis disclosed 17 out of 270 (6.3%) sites reusing the same `state` values. Table 3 in the last column shows the number of discovered test cases with constant `state` parameter for each classification. 16 out of 17 (94.1%) were found vulnerable to the CSRF against `redirect-uri`. A manual analysis showed that the use of a popup-based login prevented the completion of OCSRF attack.

The presence of a constant `state` value does not provide any additional protection to the OAuth flow as a malicious user can easily assess the existence of a "state reuse" vulnerability and include the same unchanged parameter in every attack attempt. Finally, a web application was classified as not vulnerable by the crawler (Table 3, classification 7).

Popup-Based Login. Some sites open a popup window during the login process. The developed crawler correctly handles the opening of multiple browser windows, switching the control from one to the other. Selenium has the capability to check if a secondary window is opened or closed and deal with it properly.

In some cases, the usage of a popup provides unintended protection against OCSRF attack. As an instance, we discovered one application which is not exploitable. In that case, a popup window is opened when the «Login with Facebook» button is clicked. Our crawler correctly extracts the authorization response generated for the attacker. When the URL is opened within the victim's browser session, the redirection endpoint on the target site is reached. The page response contains a few lines of JavaScript in which a function of the `window.opener` object is invoked. Since the attack URL is called directly from the address bar of the victim's browser and there is no `opener` window, an error is generated and the attack would not be completed.

However, this login architecture cannot be considered as an effective OCSRF mitigation because it does not prevent the attack from being executed with other techniques. For instance, the domain with the constant `state` appears to be vulnerable to a specifically crafted attack using a POST request. When the

login popup calls the JavaScript function in the main window, a script generates a POST request to an internal endpoint, providing the authorization code as a body parameter. The client subsequently continues the flow, contacting the authorization server to receive a valid access token. To bypass the error and complete the attack, it is sufficient to replicate the POST request using a form from a domain which is under the attacker's control. Therefore, popup-based logins do not always prevent our crawler from successfully performing attacks. We found evidence of several sites using this access strategy and many of them were exploited successfully using simple techniques.

6.5 Limitations

Some technologies built specifically to detect bots and crawlers and to interfere with their operation. We found evidence of several protections implemented by the sites tested to prevent automated login and browsing, such as CAPTCHA and similar human verification systems. An attack could fail due to the presence of a properly implemented OCSRF protection or because of the sporadic intervention of a bot detection system. However, this does not undermine the presented results as they indicate a notable lower bound for vulnerable sites.

The performed tests were not exempt from false positives. Our analysis revealed that marker information is sometimes present even if the login was not performed correctly. We verified all successful attacks by manual analysis in order to avoid including false positives which were mistakenly considered as successful in our automated crawler. Another source of errors in testing is the occurrence of temporary service unavailability. Even a highly available system has periods of downtime, for instance due to system failures, bad network conditions, or scheduled maintenance. We manually assessed the presence of these classification errors. For all the reasons outlined above, the sites classified as vulnerable by the crawler do not represent a comprehensive list but only a reasonably lower bound for the number the vulnerable sites in our analyzed candidate sites including high profile sites. This indicates the requirements of OCSRF countermeasures and the significance of the implementation mistakes which have been captured through our carefully designed attack strategies.

7 Mitigation

The OAuth 2.0 standard clearly states that developers must implement CSRF protection, by using a value that binds the authorization request to the browser session. For this purpose, the use of the state parameter is strongly recommended. The empirical evidence gathered in our work suggests that still today many OAuth implementations are vulnerable due to the absence of the state value (13%). Even when the parameter is correctly included inside the authorization URI, often it is not properly handled and validated (27%). Additionally, more than 5% of the applications tested reuse the same constant string.

Lack of adherence to the standard leaves a significant portion of websites using the OAuth 2.0 flow vulnerable to OCSRF attacks. Undeniably, identity providers have the responsibility to request the inclusion of suitable security measures. In Facebook Login the state parameter is not mandatory, and the flow works correctly also without it. The provided documentation does not help developers understand the importance of this security countermeasure and the absolute need to introduce it. At the time of writing, the examples provided do not mention that the parameter must be a random string. It is not specified how its value should be generated and there are no details about the validation process. The state value used in the practical examples is `"state123abc"`, which is also misleading as it does not help developers understand the need to make the parameter not guessable. The documentation provided by Facebook is not sufficient for a developer to build a working and secure login flow.

Alternative mitigations to OCSRF attacks involve the analysis of HTTP headers. Li et al. [14] proposed a technique based on the analysis of the Referer header field. Their strategy involves the introduction of an additional validation that must be performed by the relying party. When the client application receives the authorization response from the identity provider, it must analyse the Referer header. If the address contained in the field belongs to the relying party or to the identity provider domain, the authorization response is considered legitimate, otherwise it is discarded. The technique allows preventing the execution of OCSRF initiated from domains under the control of an attacker.

This mitigation was used inside a browser extension named OAuthGuard [15], a vulnerability scanner developed with the aim of providing real-time protection against common OAuth vulnerabilities to end-users. Even if this solution is technically valuable and relatively easy to implement, its real impact in protecting against the perils of OCSRF attacks is directly related to the number of users who employ it. From the perspective of a developer who is made aware of the threats associated with OCSRF attacks, focusing on generating a secure flow that involves the use of the `state` parameter represents probably still the best solution.

8 Conclusions

Our work is mainly focused on the analysis of the CSRF attack against `redirect_uri`, a well-known and documented OAuth 2.0 vulnerability. Our security assessment revealed that many actual implementations of OAuth-based SSO services are vulnerable to the considered attack. The reason behind the prevalence of this class of vulnerabilities is related to the complexity of implementing effective mitigations and to the absence of tools to reliably detect the threats. As a future work, we plan to test similar strategies also for other OCSRF attacks.

We designed a wide range of different, including novel, attack strategies, considering different possible implementation weaknesses and the state of the victim's browser at the time of the attack. Our analysis showed that several

enabling factors influence the feasibility of the attack and play a major role in preventing it or increasing its chances of success, augmenting the overall risk. We inspected several under-explored aspects of the vulnerability, trying to cover different areas of interest, and to expand our knowledge and understanding about the impact of the attack in different scenarios. The large number of considered test cases helped us to discover numerous well-hidden vulnerabilities and implementation mistakes. We conducted a large-scale analysis based on the approach presented, to assess the presence of OCSRF vulnerabilities in more than 300 sites implementing the Facebook Login flow. More than a third of them were found vulnerable to at least one of the designed attack scenarios. This result demonstrates that this security threat still represents a critical problem for OAuth-based authentication systems and that it probably deserves more attention from researchers and developers.

References

1. Bai, G., et al.: Authscan: automatic extraction of web authentication protocols from implementations. In: NDSS (2013)
2. Bansal, C., Bhargavan, K., Delignat-Lavaud, A., Maffeis, S.: Discovering concrete attacks on website authorization by formal analysis 1. J. Comput. Secur. **22**(4), 601–657 (2014)
3. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 75–88 (2008)
4. Calzavara, S., Focardi, R., Maffei, M., Schneidewind, C., Squarcina, M., Tempesta, M.: {WPSE}: fortifying web protocols via browser-side security monitoring. In: 27th {USENIX} Security Symposium ({USENIX} Security 2018), pp. 1493–1510 (2018)
5. Farooqi, S., Zaffar, F., Leontiadis, N., Shafiq, Z.: Measuring and mitigating oauth access token abuse by collusion networks. In: Proceedings of the 2017 Internet Measurement Conference, pp. 355–368 (2017)
6. Fett, D., Küsters, R., Schmitz, G.: SPRESSO: a secure, privacy-respecting single sign-on system for the web. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1358–1369. ACM (2015)
7. Fett, D., Küsters, R., Schmitz, G.: A comprehensive formal security analysis of OAuth 2.0. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1204–1215. ACM (2016)
8. HackerOne: Hackerone bug bounty platform (2020). https://www.hackerone.com/
9. Hardt, D.: The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012. http://www.rfc-editor.org/rfc/rfc6749.txt. http://www.rfc-editor.org/rfc/rfc6749.txt
10. Homakov, E.: The most common OAuth2 vulnerability. His Blog at (2012)
11. Kerschbaum, F.: Simple cross-site attack prevention. In: 2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007, pp. 464–472. IEEE (2007)
12. Li, F., et al.: You've got vulnerability: exploring effective vulnerability notifications. In: 25th {USENIX} Security Symposium ({USENIX} Security 2016), pp. 1033–1050 (2016)

13. Li, W., Mitchell, C.J.: Security issues in OAuth 2.0 SSO implementations. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 529–541. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13257-0_34
14. Li, W., Mitchell, C.J., Chen, T.: Mitigating CSRF attacks on OAuth 2.0 and OpenID connect. arXiv preprint arXiv:1801.07983 (2018)
15. Li, W., Mitchell, C.J., Chen, T.: Oauthguard: protecting user security and privacy with OAuth 2.0 and OpenID connect. In: Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop, pp. 35–44 (2019)
16. Lodderstedt, T., Bradley, L.F.: draft-ietf-oauth-security-topics-15 (2020). https://tools.ietf.org/html/draft-ietf-oauth-security-topics-15
17. Mirheidari, S.A., Arshad, S., Onarlioglu, K., Crispo, B., Kirda, E., Robertson, W.: Cached and confused: web cache deception in the wild. In: 29th {USENIX} Security Symposium ({USENIX} Security 2020), pp. 665–682 (2020)
18. Mladenov, V., Mainka, C., Schwenk, J.: On the security of modern single sign-on protocols: second-order vulnerabilities in openid connect. arXiv preprint arXiv:1508.04324 (2015)
19. OAuth.net: User authentication with OAuth 2.0 (2020). https://oauth.net/articles/authentication/. Accessed 30 July 2020
20. Pai, S., Sharma, Y., Kumar, S., Pai, R.M., Singh, S.: Formal verification of OAuth 2.0 using alloy framework. In: 2011 International Conference on Communication Systems and Network Technologies, pp. 655–659. IEEE (2011)
21. Shernan, E., Carter, H., Tian, D., Traynor, P., Butler, K.: More guidelines than rules: CSRF vulnerabilities from noncompliant OAuth 2.0 implementations. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) DIMVA 2015. LNCS, vol. 9148, pp. 239–260. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20550-2_13
22. Stock, B., Pellegrino, G., Rossow, C., Johns, M., Backes, M.: Hey, you have a problem: on the feasibility of large-scale web vulnerability notification. In: 25th {USENIX} Security Symposium ({USENIX} Security 2016), pp. 1015–1032 (2016)
23. Sudhodanan, A., Carbone, R., Compagna, L., Dolgin, N., Armando, A., Morelli, U.: Large-scale analysis & detection of authentication cross-site request forgeries. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 350–365. IEEE (2017)
24. Sumongkayothin, K., Rachtrachoo, P., Yupuech, A., Siriporn, K.: OVERSCAN: OAuth 2.0 scanner for missing parameters. In: Liu, J.K., Huang, X. (eds.) NSS 2019. LNCS, vol. 11928, pp. 221–233. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36938-5_13
25. Sun, S.T., Beznosov, K.: 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, pp. 378–390 (2012)
26. Lodderstedt, T.: OAuth 2.0 threat model and security considerations. RFC 6819, RFC Editor, January 2013. https://www.rfc-editor.org/rfc/rfc6819.txt. https://www.rfc-editor.org/rfc/rfc6819.txt
27. Wang, D.Y., Savage, S., Voelker, G.M.: Cloak and dagger: dynamics of web search cloaking. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 477–490 (2011)
28. Wang, R., Chen, S., Wang, X.: 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, pp. 365–379. IEEE (2012)

29. Wang, R., Zhou, Y., Chen, S., Qadeer, S., Evans, D., Gurevich, Y.: Explicating SDKS: uncovering assumptions underlying secure authentication and authorization. In: 22nd {USENIX} Security Symposium ({USENIX} Security 2013), pp. 399–314 (2013)
30. Yang, R., Li, G., Lau, W.C., Zhang, K., Hu, P.: Model-based security testing: an empirical study on OAuth 2.0 implementations. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 651–662 (2016)
31. Zhou, Y., Evans, D.: SSOScan: automated testing of web applications for single sign-on vulnerabilities. In: 23rd {USENIX} Security Symposium ({USENIX} Security 2014), pp. 495–510 (2014)