Computers & Security 121 (2022) 102859

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Practical attacks on Login CSRF in OAuth

Elham Arshad[*], Michele Benolli, Bruno Crispo

University of Trento, Italy

A R T I C L E I N F O

Article history:
Received 22 December 2021
Revised 8 June 2022
Accepted 23 July 2022
Available online 26 July 2022

Keywords:
OAuth
OpenID Connect
Request Forgery
CSRF
Login CSRF
state parameter
Custom HTTP Headers

A B S T R A C T

OAuth 2.0 is an important and well studied protocol. However, despite the presence of guidelines and best practices, the current implementations are still vulnerable and error-prone. This research mainly focused on the Cross-Site Request Forgery (CSRF) attack. This attack is one of the dangerous vulnerabilities in OAuth protocol, which has been mitigated through `state` parameter. However, despite the presence of this parameter in the OAuth deployment, many websites are still vulnerable to the OAuth-CSRF (OCSRF) attack. We studied one of the most recurrent type of OCSRF attack through a variety range of novel attack strategies based on different possible implementation weaknesses and the state of the victim's browser at the time of the attack. In order to validate them, we designed a repeatable methodology and conducted a large-scale analysis on 395 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. Based on our experiment, there was a significant rise in the number of OCSRF protection compared to the past scale analyses and yet over 25% of sites are exploitable to at least one proposed attack strategy. Despite a standard countermeasure exists to mitigate the OCSRF, our study shows that lack of awareness about implementation mistakes is an important reason for a significant number of vulnerable sites.

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 Ed. (2018) and OAuth 2.0 Security Best Current Practice Lodderstedt et al. (2020) 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.

As reported by Sudhodanan et al. (2017) CSRF vulnerabilities related to authentication and identity management services are extremely pervasive, even among the top-ranked domains. The paper is mainly focused on a specific vulnerability, the CSRF attack against the `redirect_uri` Ed. (2018), since it is 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 Barth et al. (2008) to the complete account takeover Homakov (2012). 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 and Google 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 is based on the wrong assumptions that those scenarios were not significant.

* Corresponding author.
E-mail addresses: elham.arshad@unitn.it (E. Arshad), michele.benolli@gmail.com (M. Benolli), bruno.crispo@unitn.it (B. Crispo).

https://doi.org/10.1016/j.cose.2022.102859
0167-4048/© 2022 Elsevier Ltd. All rights reserved.

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 Ed. (2018); Lodderstedt et al. (2020) in helping OAuth developers to mitigate implementation mistakes.
- We designed a repeatable methodology and conducted an automated analysis on 395 high-ranked sites of two representative IdP: Facebook and Google 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.
- We analyzed other CSRF mitigation for all the implemented attack scenarios and tested the impact of the attacks on mitigation, showing how inconsistent they are in different situations.

2. Background

This work focuses on a specific 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 OpenID Connect, that is, an authentication protocol and OAuth, that is, an authorization protocol and their implementations as Single Sign-on(SSO) in the wild. Threat model for CSRF attack and its impact are described as well.

2.1. OAuth

OAuth is an open standard for delegated authorization, commonly used to provide to third-party websites or applications limited access to specific services. The protocol allows users to authorize access to their private resources without sharing their credentials.

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.

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).

2.2. OpenID Connect

OpenID Connect allows clients of all types, e.g. Web-based and JavaScript clients, to launch sign-in flows and receive the basic information about the identity of authenticated users performed by an Authorization Server.

While OAuth 2.0 is about the resource accessibility through delegating authorization, OpenID Connect is about user authentication. Its purpose is to retrieve and store authentication information about the end users. OpenID Connect implements authentication as an extension to the OAuth 2.0 authorization process by placing the openid scope value in the Authorization Request. Information about the authentication is in a JSON Web Token (JWT) called ID Token, containing claims about the logged user and additional metadata (such as name or email address). Recordon and Fitzpatrick (2007)

According to OpenID Connect specifications, a string value, called nonce, is used to associate a client session with an ID Token. Nonce is a random string value generated by clients and originates from the authorization request. And if the nonce is considered, it will be included in the token. It helps the client to mitigate replay attacks by validating the token against the initial authorization request.

2.3. Single Sign-on Implementations

Single sign-on (SSO) is a mechanism that permits a user to use one set of login credentials (e.g., name and password) to access multiple applications. With SSO,the application or website that the user is trying to access relies on a trusted third party to verify that users are who they claim to be. Regarding to OpenID Connect and OAuth protocols, there are two types of SSO implementations in the wild.

In the former one, many identity providers (IdP) such as Facebook and Twitter use only OAuth to handle both authentication and authorization. However, OAuth is designed as an authorization protocol, but many implementations of OAuth are being deployed for web single sign-on, and thus authentication. This has led many developers and identity providers to conclude that OAuth is itself an authentication protocol and to use it as such. In this research, we take Facebook as the example of this type of SSO implementation.

In the later one, many identity providers such as Google and Microsoft implement both OpenID Connect and OAuth for handling both authentication and authorization. Since OAuth is an authorization protocol and does not handle user authentication. However, authentication protocols can be built on top of it by simply laying on top of the OAuth 2.0 protocol and it adds an additional token to the flow called an ID token for securely transmitting information between parties as an JSON object.OAuth.net (2020). In this research, we take Google as the example of this type of SSO implementation.

2.4. 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 Barth et al. (2008). There has been several studies Li and Mitchell (2014); Shernan et al. (2015); Sudhodanan et al. (2017); Yang et al. (2016) 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.

2.5. Mitigation against CSRF attacks

There are some mechanisms that a site can use to defend against CSRF attacks OWASP (2021b); verifying the origin with standard headers, synchronizer token pattern, SameSite Cookie Attribute, double submit cookies, and use of custom request headers, to name but a few. As we study CSRF in Single Sign-On through a web attacker, we consider the two effective defense mechanisms in this manner: validating a CSRF token to all state-changing requests(`state` parameter, in case of SSO), and including additional headers with XMLHttpRequest, as these two defense mechanisms were used repeatedly throughout our large dataset. These mechanisms are in use on the web today, yet not entirely satisfactory. However, we observed a small fraction of cases using other defense mechanisms which was effective to defend against some of the proposed attack scenarios.

2.5.1. State parameter

According to OAuth 2.0 specification Hardt (2012), 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 Yang et al. (2016).

2.5.2. Custom HTTP Headers

Sites implementing AJAX interfaces can defend against CSRF by setting a custom header via XMLHttpRequest (e.g., X-Requested-With) to all state-modifying requests. It has been suggested that the presence of the X-Requested-With header ensures that the request originated from a trusted domain. For example, to defend against login CSRF, the site must send the user's authentication credentials to the server via XMLHttpRequest. In this way, the site validates the headers that guarantee the integrity of the request and rejects all state-modifying requests that are not accompanied by the header. Sudhodanan et al. (2017)

Since, in this research, none of the attackers have control over the origin of the sites, custom HTTP headers are an effective protection mechanism against CSRF attacks. However, custom headers can only be used by sites implementing all their security-sensitive operations via JavaScript. In our experiment, we show that despite implementing this custom header, there are sites that are still vulnerable to Login CSRF.

2.5.3. User Interaction

The proposed defenses against CSRF attack do not require any user interactions. However, a crucial feature of CSRF is that web applications cannot recognize whether or not a request is intentionally issued by the user; So involving the user to perform an additional steps before accepting the request could prevent unauthorized operations. Examples of these steps that can act as strong CSRF defense when implemented correctly are as follows: 1) user Re-authentication, 2) One-time Token, 3) solving CAPTCHA challenges. Even though this type of defense is a very effective against CSRF, it can create an unpleasant impact on the user experience and for this reason, only a small fraction of sites implement it, as showing in our result in section 6.4.

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`Ed. (2018), 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.

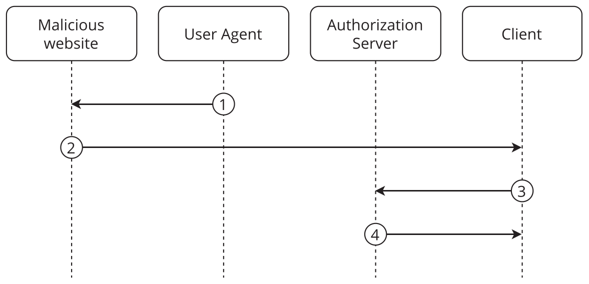


Fig. 1. Main steps involved in the OCSRF attack against `redirect_uri`

3.1. Threat model

Fig. 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.

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 Homakov (2012). 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 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. Bansal et al. (2014); Fett et al. (2016); Pai et al. (2011); Rahat et al. (2021); Wang et al. (2013)) 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. Empirical studies try to fill this gap by looking for vulnerabilities in the wild.

Many empirical works have been done on the security of OAuth-SSO (e.g., Bai et al. (2013); Farooqi et al. (2017); Fett et al. (2015); Mladenov et al. (2015); Singh et al. (2022); Sun and Beznosov (2012); Wang et al. (2012); Zhou and Evans (2014)) either by developing web-based tools or evaluating the risk in real-world implementations. In the following, we illustrate the papers covering OAuth and CSRF attacks. We categorize these related works in three groups: CSRF mitigation, protection tools, security analysis.

4.1. CSRF mitigation

There are some researches focusing on mitigations against CSRF(e.g. Li et al. (2018); Likaj et al. (2021)).

Li et al. Li et al. (2018) involve the analysis and validating Referer header field performed by the relying party. The technique allows preventing the execution of OCSRF initiated from domains under the control of an attacker. They propose a attack model and how this mitigation is effective in some cases. However, in our paper, we test a larger portion of sites automatically, including more mitigations in our results.

In Likaj et al. (2021), the authors provide a broad security evaluations of CSRF mitigations throughout the popular web frameworks. By dissecting CSRF attacks, they identify 16 existing defence mechanisms from literature and non-literature resources which considered in four distinct categories; Origin checks, Request Unguessability, SOP for Cookies, User Intention. They discover a few vulnerabilities in some of the web frameworks which allow the attacker to bypass the CSRF defense. They show that even though CSRF defenses have implemented, much of their correct and secure implementation depends on developers awareness about CSRF attacks and specific behaviors of the implementations. However, they analyze and evaluate the frameworks, web languages and current documentation to assess the CSRF attack in general. Unlike our research, not only their evaluation is not fully automated, but also they do not consider all the different factors in the real scenarios which the victim might face in the wild.

4.2. Protection tools

Another thread of research Calzavara et al. (2018); Li et al. (2019) focuses on designing and implementing browser-side tools to protect user against web attacks regarding the OAuth protocol.

Calzavara et al. Calzavara et al. (2018) design and implement 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.

The authors in Li et al. (2019) develop a browser extension named OAuthGuard with the aim of providing real-time protection against common OAuth vulnerabilities to end-users. However, the real impact of these tools against OCSRF is highly depended on the number of users who employ it.

4.3. Security Analysis

There are some papers with an approach similar to ours Li and Mitchell (2014); Shernan et al. (2015); Sudhodanan et al. (2017); Sumongkayothin et al. (2019); Yang et al. (2016). In the following, we provide a detailed explanation about these works and the gaps we address in this paper.

Li and Mitchell Li and Mitchell (2014) analyze the security of SSO implementations based on OAuth 2.0. Regarding the CSRF attack against the `redirect_uri`, authors find a significant fraction of clients are not implementing any countermeasure. Unlike our methodology, the detected security issues were manually inspected, leading to the generation of several case studies and yet they do not consider all the victim browser status in their assessment.

Sumongkayothin et al. present OVERSCAN Sumongkayothin et al. (2019), 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 and the lack of automation makes the inspection process extremely time consuming and the limited size of the resulting sample. In our tool, all actions, including login and attacks, are automated and implemented without the user interactions, leading to a greater resulting sample and the generalization of more patterns in the wild.

Yang et al. Yang et al. (2016) 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. Shernan et al. (2015) 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.Sudhodanan et al. (2017) 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 use 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 separate the environment in which attacker and victim operate. In place of performing the attack only in a clean browser session, we also perform tests in presence of visitor and authorized cookies; which is not considered in their analysis.

4.4. Gaps covered by our tool

Compared to related works, we identify three different types of login links possible through OAuth protocol (direct link, internal link and button link). We implement and automate the login part

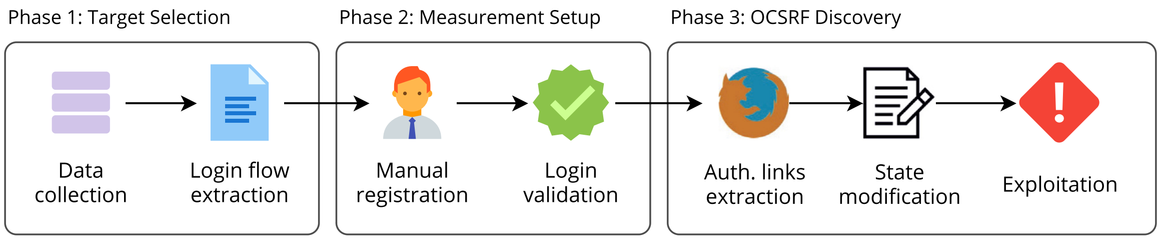


Fig. 2. Abstract view of OCSRF detection methodology.

carefully. In addition, our tool do not have the limitation of language or location of the site.

Our tool considers many more implementation parameters and factors presents in real-world implementations than existing tools, and it do not remove any site (because of presence of state parameter). The tool works with a larger number (15) of different attack scenarios compared to existing tool, some of these scenarios are completely new. The scenarios are all automated and implemented without the user interactions, which assessed to find more vulnerable sites. Unlike previous works, we analyze the impact of these attack scenarios on other CSRF mitigations and find that these factors could easily affect and disable mitigations.

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.

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 techniqueMirheidari et al. (2020) 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.

1. **No `state`.** The link is sent unaltered to the victim if the original link does not contain a `state`.

2. **Empty `state`.** The `state` value is replaced with an empty string.
3. **Lack of `state`validation.** The value of the `state` is replaced with a randomly generated string.
4. **Unlinked `state`.** The link including `state` is sent unaltered to the victim.
5. **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 Wang et al. (2011), 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 techniquesLi et al. (2016); Stock et al. (2016).

We contacted the vulnerable sites' owners through whether the support/security email or submission a new request in their sites. We explained the details of the OCSRF attack on their sites and how it could endanger the security of their users and provided some references regarding to CSRF attack. Some sites (about 9%) have replied asking more details about the vulnerability and tried to fix the problem immediately. After one month of our report, we checked the rest (the ones did not reply), only a few sites fixed the vulnerability after our report.

Additionally, we tried to disclose the vulnerabilities to those sites for which a centralized reporting system such as HackeroneHackerOne (2020) 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 conducted a large scale analysis implementing all above attack scenarios to test OCSRF attacks in the wild. In this study we discuss the measurement results of each attack strategy with different victim browser status. We managed to answer the following research questions:

(Q1) What is the popularity of OCSRF vulnerabilities on high profile and popular sites?
(Q2) Can OCSRF vulnerabilities be exploited in victim browser with unauthorized and authorized cookies?
(Q3) What are the most exploitable OCSRF attack strategies and test cases?
(Q4) Could the victim browser status be used to discover OCSRF vulnerabilities?
(Q5) What are the most popular implementation mistakes?
(Q6) How do the current mitigation mechanisms against OCSRF respond through the attack scenarios? Are they satisfactory enough to protect the victim?

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 we considered two types of SSO implementation, we targeted the most popular ones, example of each, Google and Facebook, in web applications. The crawler discovered 624 sites with Facebook login, Google login or both. 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 for Facebook and 284 for Google 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 with Facebook IdP and 117 out of 284 (%41.2) sites with Google IdP to be exploitable by at least one attack scenario. 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

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

Attack	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%)

Table 2
Number of exploitable sites in Google by OCSRF for each attack scenario

Attack	No cookies (a)	Visitor cookies (b)	Auth. cookies (c)	All
0	40 (14%)	60 (21.1%)	24 (8.4%)	60 (21.1%)
1	29 (10.2%)	32 (11.3%)	13 (4.5%)	37 (13%)
2	26 (9.1%)	31 (10.9%)	14 (4.9%)	33 (11.6%)
3	35 (12.3%)	48 (17%)	25 (8.8%)	50 (17.6%)
4	29 (10.2%)	32 (11.3%)	13 (4.6%)	36 (12.7%)
Total	80 (28.2%)	111 (39%)	50 (17.6%)	117 (41.2%)

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 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 6.2 in detail. The overall results of the crawler are summarized as follow:

1. The first category, 44 out of 314 (14.0%) sites with Facebook IdP and 65 out of 284 (22.9%) sites with Google IdP do not use `state`, which shows a significant increase in utilization of `state` compare to past large scale analysesBarth et al. (2008); Kerschbaum (2007); Li et al. (2018). It is worth mentioning that absence of `state` does not guarantee success exploitation of OCSRF as our crawler found 41 out of 44 (93.1%) with Facebook IdP and 60 out of 65 (92.3%) with Google IdP are exploitable, we will discuss case studies in 6.2.
2. The second category, 270 out of 314 (85.9%) sites with Facebook IdP and 219 out of 284 (77.1%) sites with Google IdP are using `state`. Although, this indicates a significant increase in OCSRF protection compared to the past studiesBarth et al. (2008); Kerschbaum (2007); Li et al. (2018, 2019), our crawler detected 73 out of 270 (27.0%) with Facebook IdP and 57 out of 219 (26%) with Google IdP exploitable sites utilizing different test cases. This high number indicates the complexity of OCSRF protection implementation. We will discuss the case studies in details in section 6.5.

Attack Strategies Table 1 for Facebook IdP and Table 2 for Google IdP 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% for Facebook and 17.6% for Google) in all victim browser status. In this attack strategy, as previously described in 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 in both SSO types which means some sites mistakenly accept the authorization link with an empty `state` value.

Attack Scenarios Since visitor cookie is the most vulnerable status which makes highest success rate (20.4% with Facebook IdP and 17.6% with Google IdP) and attack strategy 3: Unlinked `state` is the most effective attack strategy, attack scenario 3b has the highest detection rate. Our crawler detected 63 out of 270 (20.1%) sites with Facebook IdP and 48 out of 219 (21.9%) with Google IdP to be exploitable with it. Attack scenarios 1c and 2c had the lowest detection rates, most probably because targeted sites do not accept new OAuth login when user is authenticated.

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 cookieSudhodanan et al. (2017) or only with visitor cookieYang et al. (2016). In this research, our crawler was able to detect 23 out of 114 (20.2%) with Facebook IdP and 37 out of 117 (31.6%) with Google IdP more exploitable OCSRF cases compared to test case a: No cookies through utilizing different browser status and 9 out of 114 (7.9%) with Facebook IdP and 6 out of 117 (5.1%) with Google IdP 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 for Facebook and Table 2 for Google, 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 sites that unexpectedly were vulnerable only in this specific test cases, which would be discussed in 6.2 and 6.2.

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 in Facebook and 65 out of 284 (22.9%) of sites in Google do not set `state`. 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.

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 3 and Table 4. Each row represents one pattern w.r.t different test cases (1a,1b,etc.). A filled circle in each entry indicates successful ex-

Table 3
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

Table 4
Classification of exploitable sites in Google by OCSRF - The first category of candidates (with Absence of `state` parameter)

#	0a	0b	0c	Sites
1	●	●	●	17 (26.1%)
2	●	●	○	23 (35.3)
3	○	●	○	13 (20%)
4	○	●	●	7 (10.7%)
5	○	○	○	5 (7.6%)
Total	40	60	24	65

ploitation. The `Sites` column shows the total number of sites which have been found exploitable via the indicated pattern in corresponding row. Take Facebook for example, 3 out of 44 sites were not exploitable to any of attack scenarios, so on. While only 17 sites are vulnerable to all three attack scenarios, 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 the sites in the first category of candidates, without `state` parameter. Unexpectedly, 3 sites with Facebook IdP and 5 sites with Google IdP were not exploitable. In this regard, some 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, OAuth specification for both Facebook and Google do not 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`.

However, other sites 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 6.6.2.

Presence of `state`

Presence of the `state` does not mitigate OCSRF vulnerabilities. We summarized each exploitable pattern which was observed during our experiment on sites in our candidate set in Table 5 for Facebook and Table 6 for Google. About 73% of sites are not exploitable to any of the proposed attack scenarios. 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 sites in Facebook and 164 sites in Google 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. Custom HTTP Headers

In our database, 159 sites out of 395 (40%) implemented their login actions via XMLHttpRequest containing the X-Requested-With header. As mentioned, setting this custom header to requests defend against Login CSRF attack, however, we found that 50 sites out of 159 sites (31.4%) were still vulnerable.

Surprisingly, the sites did not set this custom header in all attack scenarios and had different behavior towards each attack. As we divided the candidates to two categories (no state and with state), out of 159 sites, 23 sites set no state parameter (first category) and 136 sites set the state parameter (second category). Out of 15 proposed attack scenarios, 3 ones applies to the first category and other 12 applies to the second category. With all consideration, if 159 sites implement this defense in all attack scenarios, there should be 1,701 occurrences. Instead, there were 1,063 occurrences in total. All 109 non-vulnerable sites behaved consistently in all attack scenarios and implemented this defense correctly and the custom header was present in all test cases. Presenting occurrences for each victim's browser status are as follows: 254 for no cookie (a), 280 for visitor cookies (b), and 529 for authorized cookies(c). The custom header mostly set when the victim is already logged in and the authorized cookies is present.

On the other hand, 50 vulnerable sites were inconsistent towards each attack scenario and they did not include the custom HTTP header in every test case. Table 7 shows the number of vulnerable sites for each test case, representing a great difference in various victim browser status. Most of them (42 out of 50 sites) are still vulnerable in visitor cookies status (b) and the least number (27 out of 50 sites) belongs to authorized cookies status. Comparing the attack strategies, the third one (Unlinked state) is the most successful one among the attack strategies. However, 12 out of 50 sites implemented the custom header in all the test cases and were still vulnerable in all situations. It is worth mentioning that 7 out of 12 sites belongs to the first category (no state).

In some cases, the custom header was included after the attack was taken successfully (in case of (a) and (b) victim browser status). As shown in classification 4,13,18 in Table 6, classification 4,13 in Table 5 and classification 2 in both Table 4 and 3, the sites are more vulnerable in (a) and (b) victim browser status. In the case of authorized cookies browser status (c), the custom header set in both situations, before and after the attack is taken, and yet this mitigation was not effective against OCSRF, as shown in classification 2,3,8,6,15 in 6, in classification 2,3,9,14 in 5, in classification 1,3 in 3 and in classification 1,4 in 4.

Generally, the custom HTTP header is more implemented and also more effective when the user is already logged in (authorized cookies status) compared to other victim browser status. So, it is one of the reasons the number of vulnerable sites in authorized cookies status is much lower than in the other two victim browser status.

6.4. User Interaction

This mitigation performs in through some techniques (e.g., re-authentication, CAPTCHA or One-time token). Based on our observation in the dataset, only 25 out of 395 sites (6%) utilized this mitigation to prevent CSRF attack; Only one site performed CAPTCHA and others (24 sites) went through re-authentication technique for this type of defense. Any site without OAuth service, utilizing a successful re-authentication technique against CSRF mitigation, redirects the user to the main login page of the site. How-

Table 5
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

Table 6
Classification of exploitable sites in Google 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	Const
1	○	○	○	○	○	○	○	○	○	○	○	○	164 (74.8%)	2 (11.8%)
2	●	●	●	●	●	●	●	●	●	●	●	●	10 (4.6%)	5 (29.4%)
3	○	○	○	○	○	○	●	●	●	○	○	○	7 (3.2%)	2 (11.8%)
4	●	●	○	●	●	○	●	●	○	●	●	○	12 (5.5%)	1 (5.9%)
5	○	○	○	○	○	○	○	●	○	○	○	○	3 (1.4%)	2 (11.8%)
6	○	○	○	○	○	○	○	●	●	○	○	○	4 (1.8%)	2 (11.8%)
7	○	●	○	○	●	○	○	●	○	○	●	○	4 (1.8%)	1 (5.9%)
8	○	○	○	●	●	●	●	●	●	○	○	○	2 (0.9%)	1 (5.9%)
9	●	○	○	●	○	○	●	○	○	●	○	○	2 (0.9%)	0
10	○	●	○	○	○	○	○	●	○	○	●	○	1 (0.5%)	0
11	○	●	○	○	○	○	○	○	○	○	●	○	1 (0.5%)	0
12	●	○	○	○	○	○	○	○	○	●	○	○	2 (0.9%)	0
13	●	●	○	○	○	○	○	○	○	●	●	○	1 (0.5%)	0
14	○	○	○	○	●	○	○	●	○	○	○	○	1 (0.5%)	0
15	○	●	●	○	●	●	○	●	●	○	●	●	2 (0.6%)	0
16	●	○	●	○	○	○	○	○	○	●	○	●	1 (0.5%)	0
17	○	○	○	○	○	○	●	●	○	○	○	○	1 (0.5%)	0
18	●	●	○	○	○	○	●	●	○	●	●	○	1 (0.5%)	1 (5.9%)
Total	29	32	13	26	31	14	35	48	25	29	32	13	219	17

Table 7
Number of exploitable sites with custom header for each attack scenario

Attack	No cookies (a)	Visitor cookies (b)	Auth. cookies (c)	All
0	12 (7.5%)	19 (12%)	13 (8.2%)	20 (12.6%)
1	10 (6.3%)	10 (6.3%)	7 (4.4%)	15 (3.1%)
2	7 (4.4%)	10 (6.3%)	5 (3.1%)	10 (6.3%)
3	17 (10.7%)	22 (13.9%)	10 (6.3%)	24 (15.1%)
4	10 (6.3%)	9 (5.7%)	9 (5.7%)	16 (10%)
Total	33 (20.7%)	42 (26.4%)	27 (17%)	50 (31.4%)

ever, within OAuth protocol, when the re-authentication technique occurs, the user will be redirected to the IdP(Google/Facebook) login page, requiring the Google/Facebook credentials. In our experiment, proposing 15 different attack scenarios, this technique occurred in case of (a) and (b) victim browser status and none in case of (c) authenticated cookies. This technique was successfully performed in the victim browser (in case of C victim browser status) and the reason it did not require the re-authentication by the user is that the authenticated cookies are already present is in the victim browser and as a result, the login CSRF attack in this case (c) was not successful, redirecting the victim to his own account instead of attacker's account. As shown in other mitigation against OCSRF, the mitigation provided in the sites are not consistent in all the attack scenarios and some factors (e.g., the victim browser status and fuzzing the state parameter value) changes the course of CSRF mitigation, leading a successful attack. This mitigation is no exception: While 19 sites performed this mitigation in all attack scenarios stopping the OCSRF attack successfully, 6 sites were inconsistent with utilizing this mitigation causing the sites vulnerable in some scenarios.

6.5. Case Studies

In this section, different attack scenarios 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 & 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.

A couple manually analyzed sites has been discovered to be exploitable only to attacks 1 and 4, as illustrated in Table 5, classification 8, 12, 17 for Facebook and in Table 6, classification 11, 12, 13, 16 for Google. 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. This behaviour can be clarified by following programming exemplification:

```
if state in response and state:
if not is_valid(state):
#Raise an exception and block the flow
#Continue and complete the flow
```

In Facebook SSO implementation, a couple of sites are exploitable only via attack strategy 1 but not the 4th (Refer to 5, 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 5, 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 (For Facebook in Table 5, classification 9 and 10. For Google in Table 6, classification 8 and 14). 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.

Unlinked state In attack strategy 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 for Facebook and 2 for Google, attack strategy 3 is the most successful one. More than 20% of the candidate site are vulnerable to scenario 3a, 3b, or 3c for both Google and Facebook in total. 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. For Facebook, 23 out of 114 (20.2%) and for Google, 15 out of 117 (12.8%) exploitable sites are only vulnerable to attack strategy 3 (In Table 5, classifications 3,5,7,16, and 18. In Table 6, classification 3, 5, 6 and 17). 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 with Facebook IdP and seven sites with Google IdP are vulnerable to all configurations of attack strategy 3 (Table 5, classifications 3 and Table 6, classifications 3). In total, 5 sites were incorrectly classified by the crawler in this group due to the notification message to the victim, which we considered as false positives. The sample message is shown as follows.

> The account ({attacker email}) belongs to another user ({attacker name}). You can log in as {attacker name} or cancel to stay logged in as {victim name}.

The crawler wrongly classified sites due to the presence of marker information related to the attacker's account. The attack is not blocked by default, but the above-mentioned informative message helps the victim to make the correct decision. Although not recommended, we classified the sites which show their users a similar message as secure (not vulnerable).

Lack of state Validation 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. For Facebook, 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. For Google, the numbers are 26 out of 117 (22.2%), 31 out of 117 (26.5%) and 14 out of 117 (12%) respectively. Presence of visitor cookies increases the attack success rate similar to other attack strategies.

It can be easily noticed from Table 5 and Table 6 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.

Specific Characters in generating state :

A particular case in attack strategy 2 is represented by sites that use a state consisting of a single character, or a sequence of N identical characters. In this strategy, 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 scenario 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.

6.6. Notable Observations

6.6.1. 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 æstate reuseg 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 with Facebook IdP and 17 out of 219 (7.7%) sites with Google IdP reusing the same state values. Table 5 and 6 in the last column shows the number of discovered vulnerable sites with constant state parameter for each classification for Facebook and Google respectively. 16 out of 17 (94.1%) in Facebook and 15 out of 17 in Google 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 couple of sites classified as not vulnerable by the crawler. (Table 5, classification 7 and Table 6, classification 5).

6.6.2. Other CSRF defense mechanisms

As an example, we discovered one application which is not exploitable, using the SameSite cookie attribute in addition to the state parameter OWASP (2021a) against Login CSRF. SameSite is a cookie attribute which instructs the browser on whether to send cookies along with cross-site requests. The application opens a popup window during the login process. 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. This cookie attribute can take the values Lax, Strict, or None. In the example above, the application has set the SameSite to Lax. The Lax value will be sent only in the GET request in top-level window navigations and blocks the cross-site requests in CSRF-prone request methods such as POST. It is worth mentioning that this attribute should not be replaced a CSRF Token and only implemented as an additional layer of defense. This attribute alone protects the user through the browsers supporting it and it could be bypassed easily by replicating the POST request using a form from a domain which is under the attacker's control.

In double submit cookie OWASP (2021b), a site can set a CSRF token as a cookie, and also insert it as a hidden field in each HTML form. When the form is submitted, the site can check that the cookie token matches the form token. This mechanisms is stateless, since it does not require any server side state for CSRF token. In Table 5 and Table 6, classification 4, the sites are vulnerable in all configuration, except the authorized cookies browser status (c). We inspected the HTTP traffic and discovered seven sites with presence of CSRF token in cookies, as 'xf_csrf', and in HTTP header, as 'X-CSRF-TOKEN or 'X-XSRF-TOKEN'. However, these sites used this technique inconsistently and only in authorized cookies browser status(c). In another words, when a user logs into the site, the double submit cookie implements and only effective in auth. cookie browser status(c).

6.6.3. Comparison Facebook and Google results

The result for Google and Facebook is quite similar. In the candidate sites, the flow and process of both IdPs are the same as we observed in our collected data. As follows, we provide the results for google comparing to Facebook. In both Google and Facebook login, 3b(unlinked state: [visitor cookie]) attack strategy has the highest success rate (22.2% for Google and 20.1% for Facebook) in all victim browser status. Visitor cookie is the most vulnerable status in both IdPs, which makes highest success rate compare to other attack strategies and 3b is the most effective attack strategy. In both IdPs, Test cases 1c and 2c had the lowest detection rates, most probably because targeted sites do not accept new OAuth login when the user is authenticated. More than 20% of the candidate sites in both IdPs are vulnerable to attack 3. As we concluded in the main draft, OCSRF attacks have a better chance of success in presence of visitor cookies. With all the similarities in the result for both IdPs, there are some slight differences. Based on our observation, Google mostly uses OpenID Connect + OAuth protocol, which OpenID Connect is used for authentication. The OpenID Connect protocol suggests that sites include a self-signed Nonce to protect against reply attacks. The sites should generate a fresh Nonce, save the Nonce in the browser's cookie and place Nonce in the return_to parameter of the OpenID protocol. The site should validate that the Nonce value in the return_to URL matches the Nonce stored in the cookie, after obtaining the identity assertion from the user's Identity Provider. It ensures that the OpenID protocol session completes on the same browser as it began. Among 284 sites with Google IdP, only 5 are using Nonce parameter and 4 of them are immune to all the attack scenarios. The other one is only exploitable to attack 1b, 3b, and 4b. (visitor cookie: Empty state, Unlinked state, Missing state) and surprisingly not vulnerable to 2b (visitor cookie: lack of state validation) as shown in Table 6, classification 10. The reason might be that the value for both state and nonce parameter in the authorization url is always the same and any change in state parameter would raise an error in server-side. However, our crawler only modifies the state parameter in the authorization url. So we manually tested the attack scenario 2b on this particular site; We modified both state and nonce parameters to the same value and surprisingly, the site was vulnerable to attack scenario 2b within this small change. Also, we found one site which use the constant state and immune to attack scenario 2(lack of state validation) in Table6, classification 18, Column 'Const'. The reason might be the request is using the constant state parameter and any changes also caused an error in server side. However, the site were vulnerable to other attack scenarios. In our dataset, 197 sites are deploying both Google and Facebook API, which 96 sites (48.7%) are immune to OCSRF attack in both IdPs. 38 out of 197 sites (19.3%) are vulnerable and have the same results and pattern in all the attack scenarios. The remaining sites (63 out of 197 sites) have the different pattern for all the attack scenarios. However, in this sample, the number of vulnerable sites with Google IdP is almost twice the ones with Facebook IdP, i.e., there are 25 sites which are only the Google IdP is vulnerable to OCSRF attacks and 15 sites which are only the Facebook IdP is vulnerable.

6.7. 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% for Facebook and 21% for Google). Even when the parameter is correctly included inside the authorization URI, often it is not properly handled and validated (27% for Facebook and 26% for Google). Additionally, more than 5% of the applications tested reuse the same constant string for both Google and Facebook IdPs.

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 some SSO services (e.g. Google and Facebook), 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 some IdPs 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. Li et al. (2018) 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 Li et al. (2019), 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 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 395 sites implementing two types of SSO implementations (i.e. Google and Facebook). 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 and OpenID Connect + OAuth authentication systems and that it probably deserves more attention from researchers and developers.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Elham Arshad: Conceptualization, Methodology, Validation, Investigation, Writing – review & editing. **Michele Benolli:** Software, Data curation, Writing – original draft. **Bruno Crispo:** Project administration, Supervision.

References

Bai, G., Lei, J., Meng, G., Venkatraman, S.S., Saxena, P., Sun, J., Liu, Y., Dong, J.S., 2013. Authscan: Automatic extraction of web authentication protocols from implementations. NDSS.

Bansal, C., Bhargavan, K., Delignat-Lavaud, A., Maffeis, S., 2014. Discovering concrete attacks on website authorization by formal analysis 1. Journal of Computer Security 22 (4), 601–657.

Barth, A., Jackson, C., Mitchell, J.C., 2008. Robust defenses for cross-site request forgery. In: Proceedings of the 15th ACM conference on Computer and communications security, pp. 75–88.

Calzavara, S., Focardi, R., Maffei, M., Schneidewind, C., Squarcina, M., Tempesta, M., 2018. {WPSE}: Fortifying web protocols via browser-side security monitoring. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 1493–1510.

Ed., T.L., 2018. OAuth 2.0 Threat Model and Security Considerations. RFC. https://www.rfc-editor.org/rfc/rfc6819.txt

Farooqi, S., Zaffar, F., Leontiadis, N., Shafiq, Z., 2017. Measuring and mitigating oauth access token abuse by collusion networks. In: Proceedings of the 2017 Internet Measurement Conference, pp. 355–368.

Fett, D., Küsters, R., Schmitz, G., 2015. 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. ACM, pp. 1358–1369.

Fett, D., Küsters, R., Schmitz, G., 2016. A comprehensive formal security analysis of OAuth 2.0. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1204–1215.

HackerOne. Hackerone bug bounty platform. 2020. https://www.hackerone.com/.

Hardt, D., 2012. The OAuth 2.0 Authorization Framework. RFC. http://www.rfc-editor.org/rfc/rfc6749.txt

Homakov, E., 2012. The most common OAuth2 vulnerability. Technical Report. http://homakov.blogspot.com/2012/07/saferweb-most-common-oauth2.html

Kerschbaum, F., 2007. Simple cross-site attack prevention. In: 2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007. IEEE, pp. 464–472.

Li, F., Durumeric, Z., Czyz, J., Karami, M., Bailey, M., McCoy, D., Savage, S., Paxson, V., 2016. You've got vulnerability: Exploring effective vulnerability notifications. In: 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 1033–1050.

Li, W., Mitchell, C.J., 2014. Security issues in OAuth 2.0 SSO implementations. In: International Conference on Information Security. Springer, pp. 529–541.

Li, W., Mitchell, C.J., Chen, T., 2018. Mitigating CSRF attacks on OAuth 2.0 and openID connect. arXiv preprint arXiv:180107983.

Li, W., Mitchell, C.J., Chen, T., 2019. 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.

Likaj, Xhelal, Khodayari, S., Giancarlo, P., 2021. Where we stand (or fall): An analysis of CSRF defenses in web frameworks. 24th International Symposium on Research in Attacks, Intrusions and Defenses..

Lodderstedt, Bradley, Labunets, Fett, 2020. draft-ietf-oauth-security-topics-15. Technical Report. https://tools.ietf.org/html/draft-ietf-oauth-security-topics-15

Mirheidari, S.A., Arshad, S., Onarlioglu, K., Crispo, B., Kirda, E., Robertson, W., 2020. Cached and confused: Web cache deception in the wild. In: 29th {USENIX} Security Symposium ({USENIX} Security 20), pp. 665–682.

Mladenov, V., Mainka, C., Schwenk, J., 2015. On the security of modern single sign-on protocols: Second-order vulnerabilities in openid connect. arXiv preprint arXiv:150804324.

OAuth.net, 2020. User Authentication with OAuth 2.0. Technical Report. https://oauth.net/articles/authentication/

OWASP. Cookies: HTTP state management mechanism draft-ietf-httpbis-rfc6265bis-02. 2021a. https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis-02-section-5.3.7.

OWASP. Cross-site request forgery prevention. 2021b. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

Pai, S., Sharma, Y., Kumar, S., Pai, R.M., Singh, S., 2011. Formal verification of OAuth 2.0 using alloy framework. In: 2011 International Conference on Communication Systems and Network Technologies. IEEE, pp. 655–659.

Rahat, Al, T., Feng, Y., Yuan, T., 2021. Oauthshield: Efficient security checking for oauth service provider implementations. arXiv preprint arXiv:2110.01005

Recordon, D., Fitzpatrick, B., 2007. OpenID authentication 2.0-final. Network Working Group Internet-Draft. https://openid.net/specs/openid-authentication-2_0.html

Shernan, E., Carter, H., Tian, D., Traynor, P., Butler, K., 2015. More guidelines than rules: CSRF vulnerabilities from noncompliant OAuth 2.0 implementations. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 239–260.

Singh, Jaimandeep, Naveen Kumar, C., 2022. Oauth 2.0: Architectural design augmentation for mitigation of common security vulnerabilities. Journal of Information Security and Applications 65.

Stock, B., Pellegrino, G., Rossow, C., Johns, M., Backes, M., 2016. Hey, you have a problem: On the feasibility of large-scale web vulnerability notification. In: 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 1015–1032.

Sudhodanan, A., Carbone, R., Compagna, L., Dolgin, N., Armando, A., Morelli, U., 2017. Large-scale analysis & detection of authentication cross-site request forgeries. In: 2017 IEEE European symposium on security and privacy (EuroS&P). IEEE, pp. 350–365.

Sumongkayothin, K., Rachtrachoo, P., Yupuech, A., Siriporn, K., 2019. Overscan: Oauth 2.0 scanner for missing parameters. In: International Conference on Network and System Security. Springer, pp. 221–233.

Sun, S.T., Beznosov, K., 2012. 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.

Wang, D.Y., Savage, S., Voelker, G.M., 2011. Cloak and dagger: dynamics of web search cloaking. In: Proceedings of the 18th ACM conference on Computer and communications security, pp. 477–490.

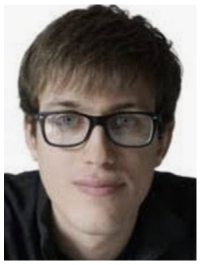
Wang, R., Chen, S., Wang, X., 2012. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In: 2012 IEEE Symposium on Security and Privacy. IEEE, pp. 365–379.

Wang, R., Zhou, Y., Chen, S., Qadeer, S., Evans, D., Gurevich, Y., 2013. Explicating sdks: Uncovering assumptions underlying secure authentication and authorization. 22nd {USENIX} Security Symposium ({USENIX} Security 13). 399–314

Yang, R., Li, G., Lau, W.C., Zhang, K., Hu, P., 2016. 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.

Zhou, Y., Evans, D., 2014. Ssoscan: automated testing of web applications for single sign-on vulnerabilities. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 495–510.

Elham Arshad is currently a PhD student at the University of Trento,Italy, where previously worked as a scientific researcher in computer science department. She received her masters degree from Sharif University of Technology, Iran, in computer science, 2016. She has over 5 years job experience as a senior developer and security specialist in high-tech companies. Her major research interests lie in web application Security and privacy and mainly focused on large-scale security measurement, using browser instrumentation and distributed crawling.



Michele Benolli received the Bachelor and Master degree (cum laude) in Computer Science from the University of Trento, Italy, in 2015 and 2020, respectively. His academic interests are mainly related to the field of web security. During his Master's, he collaborated with the DISI Security Research Group for realizing large-scale analysis of web-based attacks and automated vulnerability detection.

Bruno Crispo is Professor at the Department of Computer Science at theUniversity of Trento in Italy. His main research interest lies in the area of systemand network security and access control. In particular, he is currently workingon smartphone and mobile app security, security and privacy on the Internet of Things and behavioral biometrics.He has published more than 140 papers in scientific journalsand international conferences. He is an Associate Editor of the ACMTransactions on Privacy and Security. He is a Senior Member of IEEE.