VietNam National University

University of Engineering and Technology



Family name: Nguyen Sinh

First name: Hien

Student ID: 19021269

Unit code: INT3307\_21

Unit name: Network Security

Name of Lecturer: Nguyen Dai Tho

Student Email address: [19021269@vnu.edu.vn](mailto:19021269@vnu.edu.vn)

Dated: 10/01/2022

Ha Noi, 2022



## CROSS-SITE REQUEST FORGENCY

## ATTACKS ON OAUTH 2.0 SYSTEMS

# ABSTRACT

The OAuth 2.0 protocol is one of the most widely deployed authorization/single sign-on (SSO) and cross-site request forgency (CSRF) is one of the most important vulnerabilities of Web Application for a decade since it was first identified. In this literature, we focus on OAuth protocol, CSRF vulnerabilities, explore the real-world CSRF attack on OAuth 2.0 , and analysis the solution to defence CSRF attack.

## Introduction

Nowadays, millions of web application users in the world using their Google accounts to sign into more than one million *Relying Party* websites. This web-based single-sign on (SSO) schema is enabled by OAuth 2.0 – a web resource authorization protocol which has been adopted by major service providers. Popular social networks such as Google, Facebook and Twitter implement their APIs (Application Programming Interfaces) based on the OAuth protocol to enhance user experience of social sign-on and social sharing. By using OAuth, the website can easily manage user’s account, as well as save them from the inconvenience of creating a new account or re-typing attributes that are instead stored by identity providers and provided to relying parties as required.

However, besides the convenience of OAuth 2.0, there are many vulnerabilities occur led application developers have struggled to develop secure implementations. In this literature, we consider the extent to which a specific documented vulnerability is embodied in real-world implementations. Specifically, Cross Site Request Forgery (CSRF) attacks on OAuth systems, which may force an unsuspecting user to perform an “authorized” action without their knowledge (eg: access to strange link to transmit financial information to a third-party, modify sensitive file, etc.). The OAuth 2.0 standard cannot have ability to prevent the danger that vulnerabilities pose, and notes that both client and server “MUST implement CSRF protection” [1].

The literature is structured as follows:

* Section 2 introduces OAuth 2.0, Authorization code flow and CSRF attack.
* Section 3 describes a CSRF attack on OAuth system.
* Section 4 propose the solution and analysis it’s props and cons.
* Section 5 we draw conclusion.

## 2. Background

### **2.1. Oauth 2.0**

OAuth (Open Authorization) is an open standard for access delegation [3]. The OAuth protocol allows user to grant a web application authorized access to our data on a different application [4]. Specifically, it allows user to grant a *Relying Party* the ability to perform a set of operations on an account with an *Identity Provider* [2]. The set of permissed actions are provided by the *Identity Provider* in an API. A common application of distributing authorization via OAuth is single-sign on (SSO), which allows users to connect to multiple web services (*Relying Partys*) using one set of credentials from an *Identity Provider* [20] (e.g., logging into https://dantri.com.vn with Facebook account to comment an article).

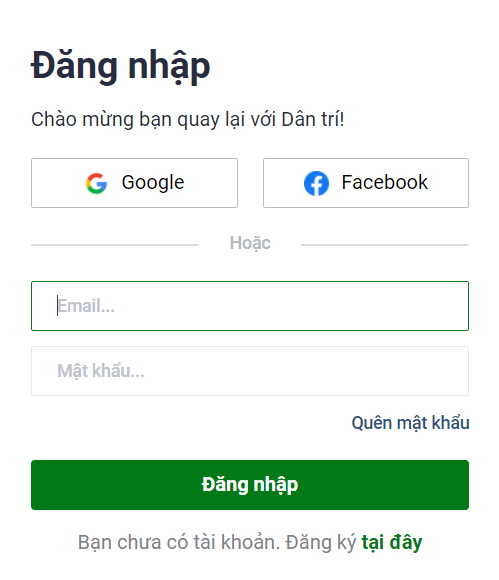


Fig. 1. OAuth in dantri.com.vn

OAuth 2.0 first authenticates the user with a username and password, followed by consent given at a permissions screen for the *Relying Party* to access the identity stored at the *Identity Provider* [1]. The protocol relies on the use of access tokens for authorizing requests made by an *Relying Party* to access data provided by an *Identity Provider* on behalf of a user [5]. OAuth 2.0 has multiple forms of authentication flows to allow for use in different scenarios such as native, web, and mobile applications [1]. The significance of the OAuth 2.0 protocol is clear, as research has shown that a growing number of web applications are relying heavily on a small set of *Identity Providers*, many of which (Google, Facebook, Twitter, Zalo, etc) employ the OAuth 2.0 protocol. Moreover, research is proposing the use of OAuth 2.0 as a important component for authentication [6].

### **2.2. Authorization code flow**

Instead of requesting authorization directly from the resource owner, the client directs the resource owner to an authorization server. The resource owner is then redirected back to the client with the authorization code which the client will capture and exchange for an access token in the background. Since this is a redirection-based flow, the client must be able to interact with the resource owner's user-agent and receive incoming requests (via redirection) from the authorization server.

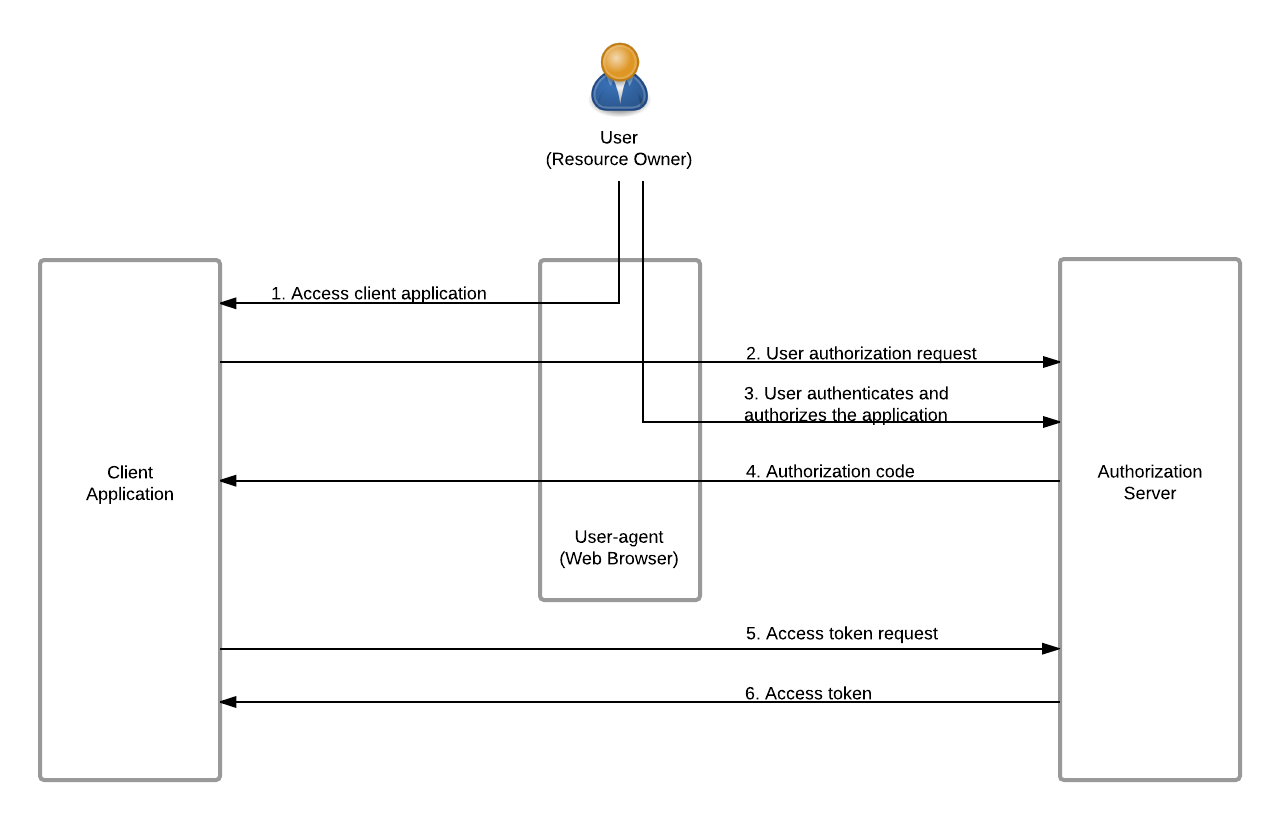


Fig. 2. Authorization code grant flow [7]

Step 1: User initiates the flow by accessing to client application through the web browser.

Step 2: Client application send authorization request to authorization server.

Step 3: The authorization server authenticates the user ( via the web broswer) and establishes wherther the user grants or denies the client’s access request.

Step 4: If the user grants access, the authorization server redirects the web browser back to the client using the redirection URI provided earlier. The redirection URI includes an authorization code and any local state provided by the client earlier.

Step 5: Then, the client requests an access token from the authorization server’s token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with authorization server. The client includes the redirection URI used to obtain the authorization code for verification.

Step 6: The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step 4. If all valided, the authorization server send back to the client with an access token and, optionally, a refresh token.

### **2.3. Cross-site Request Forgency**

In a CSRF attack, the attacker's goal is to cause an innocent victim to unknowingly submit a maliciously crafted web request to a website that the victim has privileged access to. This web request can be crafted to include URL parameters, cookies and other data that appear normal to the web server processing the request. At risk are web application that perform actions based on input from trusted and authenticated users without requiring the user to authorize the specific action. A user who is authenticated by a cookie saved in the user's web browser could unknowingly send an HTTP request to a site that trusts the user and thereby cause an unwanted action.

A general property of web browsers is that they will automatically and invisibly include any cookies used by a given domain in any web request sent to that domain. This property is exploited by CSRF attacks in that any web request made by a browser will automatically include any cookies (including session cookies and others) created when a victim logs into a website. In the event that a user is tricked into inadvertently submitting a request through their browser these automatically included cookies will cause the forged request to appear real to the web server and it will perform any appropriately requested actions including returning data, manipulating session state, or making changes to the victim's account.

In order for a CSRF attack to work, an attacker must identify a reproducible web request that executes a specific action such as changing an account password on the target page. Once such a request is identified, a link can be created that generates this malicious request and that link can be embedded on a page within the attacker's control. This link may be placed in such a way that it is not even necessary for the victim to click the link. For example, it may be embedded within an html image tag on an email sent to the victim which will automatically be loaded when the victim opens their email. Once the victim has clicked the link, their browser will automatically include any cookies used by that website and submit the request to the web server. The web server will not be able to identify the forgery because the request was made by a user that was logged in, and submitted all the requisite cookies.

Cross-site request forgery is an example of a *Confuse Deputy Attack* against a web browser because the web browser is tricked into submitting a forged request by a less privileged attacker.

CSRF commonly has the following characteristics:

* It involves sites that rely on a user's identity.
* It exploits the site's trust in that identity.
* It tricks the user's browser into sending HTTP requests to a target site.
* It involves HTTP requests that have side effects.

## 3. CSRF attack on OAuth system

## 4. Solution

## 5. Conclusion

## 6. References

[1] Hardt, D.: *The OAuth 2.0 authorization framework*. RFC 6749, RFC Editor, October 2012.

[2] Shernan, E., Carter, H., Tian, D., Traynor, P., & Butler, K. (2015). *More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations. Lecture Notes in Computer Science, 239–260.* doi:10.1007/978-3-319-20550-2\_13

[3] Wikipedia (<https://en.wikipedia.org/wiki/OAuth>)

[4] Kaur, G., Aggarwal, D.: *A survey paper on social sign-on protocol OAuth 2.0*. J. Eng. Comput. Appl. Sci. 2(6), 93–96 (2013)

[5] Jones, M., Hardt, D.: *The OAuth 2.0 authorization framework: bearer token usage*. RFC 6750, RFC Editor, October 2012.

[6] Nauman, M., Khan, S., Othman, A.T., Musa, S.U., Rehman, N.U.: *POAuth: privacy-aware open authorization for native apps on smartphone platforms*. In: Proceedings of the International Conference on Ubiquitous Information Management and Communication (2012)

[7] <https://is.docs.wso2.com/en/latest/learn/authorization-code-grant>