ELEKTROTEHNIŠKI VESTNIK 78(X): 1–2, 2011
ENGLISH EDITION

Mitigating Cross-Site Request Forgery Attacks

Matjaž Mav

University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, 1000 Ljubljana, Slovenia
E-mail: mm3058@student.uni-lj.si

Abstract. Because of accessibility, many native applications transitioned to the web. Additionally many users are using social login option on the daily bases. In this paper we will discuss various techniques for Cross-Site Request Forgery attack mitigation; specifically we will illustrate possible attack on OAuth 2.0 authorization protocol.

Key words: Cross-Site Request Forger, CSRF, XSRF, XSFR, OAuth 2.0, OpenID Connect 1.0

1 INTRODUCTION

Cross-Site Request Forgery (CSRF) is an attack where authenticated user executes unwanted action of attacker's choosing. This attacks usually target state-changing requests (for example executing unwanted bank transaction) and not compromising data secrecy. But it is also possible to attack the login process and this can lead to private data exposure [1].

In this paper, we will use CSRF as an abbreviation for Cross-Site Request Forgery. In other texts, we can notice XSRF or XSFR used as synonyms for CSRF abbreviation.

The basis for this paper is an article *Mitigating CSRF attacks on OAuth 2.0 Systems* [4] published in 2018.

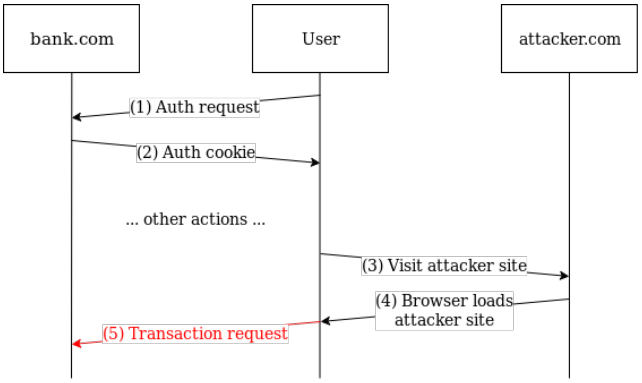

Figure 1. The simple example of a CSRF attack.

2 CSRF BACKGROUND

In this section we will go through simple CSRF attack example, where authenticated user unintentionally executes bank transaction (figure 1).

First user sends (1) authentication request to the online bank (*bank.com*) and receives back (2) authentication cookie. Then he opens seemingly friendly email and (3) clicks on the link. Link opens browser and (4) loads attacker site (*attacker.com*). Without user knowledge, this malicious site sends HTTP request to the bank and (5) create bank transaction. Because user's browser still has valid authentication cookie, the malicious HTTP request is automatically accompanied with authentication cookie.

To mitigate CSRF attacks we commonly use some combination of the following approaches:

1) The first approach uses randomly generated tokens that are sent at each HTTP request. These tokens are randomly generated on the server and then two copies are sent back to the browser. One copy is stored inside the cookie and the other one is injected into form. When the form is submitted, the server compares value stored inside a cookie with value submitted over form. If these two values are valid and match request is further processed, otherwise request is rejected [4].
2) The second approach prevent execution requests that are initiated from different origins. This can be achieved using either (1) `Referer` request header, (2) custom HTTP headers or (3) `Origin` request header [1], [4]. Each of this approaches can distinguish between same-origin or cross-origin requests.
3) The third and most recent approach is to use authentication cookie with `SameSite` attribute [2]. Cookies annotated with `SameSite` attribute are only .

The second and third approach heavily relies on the specific browser and thus cannot be completely trusted.

Received ???
Accepted ???

3 OAuth 2.0 background

In this section we will briefly explain what authorization protocol OAuth 2.0 is and how it works.

OAuth 2.0 [3] allows an *Client* to access resources protected by *Resource Server* on behalf of the *Resource Owner*. This is done by consuming access token issued by *Authorization Server*. OAuth 2.0 protocol heavily relies on browser redirects and therefore can be subject of CSRF attack.

OpenID Connect 1.0, which is an identity layer build on top of the OAuth 2.0 protocol, is commonly used for Single Sign-On (SSO) scenarios.

OAuth 2.0 has four different protocol flows [3]. Here we will only discuss one of them; that is Implicit Grant flow. The protocol for Implicit Grant flow roughly proceeds as follows [4]:

1) The user connects to the client application and clicks a social login button.
2) The browser redirects user to the authorization server with following query parameters: (1) `response_type=token` indicating Implicit Grant flow, (2) `state`, (3) `scope` of the request permissions, (4) `client_id` and (5) `redirect_uri`.
3) The authorization server first validate `client_id` and registered redirect URI with `redirect_uri` parameter. If there is a match, process continues.
4) The user enter credentials and grants permissions for the requested `scopes`.
5) The authorization server generates an `access_token` and redirects back to the `redirect_uri` following by symbol # and then by the generated `access_token`.
6) The client application stores `access_token` for later usage.

4 CSRF attack on OAuth 2.0

CSRF attack against OAuth 2.0 `redirect_uri` can allow an attacker to associates the victim's session with the attacker's `access_token`. The victim then accesses the protected resource on behalf of the attacker. This can lead to private data exposure; i.e. the user might upload private data to the attacker's account.

The attacker can achieve this by first obtains his `access_token` and then aborts the redirect flow. Then the attacker tricks the victim to execute the attacker's redirect flow, by this victim's session gets associated by the attacker's `access_token`.

Because OAuth 2.0 protocol relies on cross-origin requests to obtain an `access_token`, previously discussed mitigation approaches don't apply to this kind of attack. The exception is the first approach that uses secret token; this can be translated into `state` parameter.

In practice this mitigation approach is commonly not used or it is misused. Therefore the authors of article *Mitigating CSRF attacks on OAuth 2.0 Systems* [4] suggest another simple approach. They suggest validation of the `Referer` request header value with authorization server domain name.

5 Conclusions

According to the OWASP Top 10 report from the year 2013 [5], Cross-Site Request Forgery attacks ware ranked at number 8 of top 10 most critical web application security risks. However, in the most recent OWASP Top 10 report form year 2017, CSRF is no longer on the list.

We believe that few of the reasons for its fall from the top 10 list are:

1) More frameworks are now offering secure-by-default settings.
2) New and improved browser standards were introduced for CSRF attack mitigation.
3) Raised overall awareness for secure development.

References

[1] Adam Barth, Collin Jackson, and John C Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88. ACM, 2008.

[2] Mark Goodwin and Mike West. Same-site Cookies. https://tools.ietf.org/html/draft-west-first-party-cookies-07, Apr 2016. [Online; accessed 25. Nov. 2018].

[3] Dick Hardt. The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749, Oct 2012. [Online; accessed 25. Nov. 2018].

[4] Wanpeng Li, Chris J Mitchell, and Thomas Chen. Mitigating csrf attacks on oauth 2.0 systems. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pages 1–5. IEEE, 2018.

[5] Category:OWASP Top Ten Project - OWASP. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. [Online; accessed 24. Nov. 2018].