CS-573
Assignment 2

CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations

(Harshala Yadav)

Need for thorough guidelines to tackle CSRF Vulnerabilities[1]

Abstract:

The paper focuses on the lack of comprehensive documentation and knowledge base to implement mitigation for Cross Site Request Forgery (CSRF) vulnerabilities in OAuth 2.0 framework. The framework is one of the most widely adopted implementation to authorize users across Identity Providers and Relying parties. The paper analyzes the extent to which CSRF vulnerability exists among the top 10k ranked websites on Alexa.com – a popular web traffic ranking website. The paper found that 25% of the website which relied on the OAuth framework for authorization were vulnerable to some extent. The paper analyzes four high-profile use-cases in particular which reveal the key takeaways and the argument of this paper.

Introduction:

OAuth 2.0, an IETE standard (The Internet Standard), is one of the most widely adopted framework to authorize sharing of data from disparate domains, a true revolution in the web application industry. OAuth 2.0 provides a secure way to access application data and user accounts. The flexibility of this standard allows implementation of additional features like authentication and single sign-on.

However, every new implementation has its own security challenges to be tackled by developers. The OAuth 2.0 standard explicitly lists the possibility of a CSRF attack forcing an unsuspecting user to perform an authorized action without their knowledge. Non-compliance to the standards of the framework leads to vulnerabilities. According to research of the paper, ample documentation exists for OAuth 2.0 framework implementation. However, mitigation techniques for the CSRF vulnerability, the subject matter of this research paper, is sporadic and many a times does not conform to the OAuth 2.0 standard. The aim of this paper is the root cause analysis of this inadequate implementation for this vulnerability which remains unresolved.

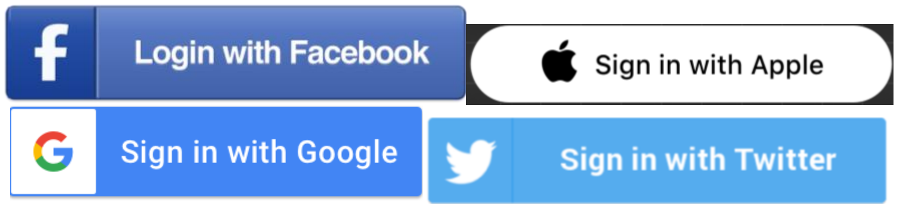
OAuth 2.0:

The primary idea of the OAuth protocol is to allow a user to grant a web application authorized access to his data on a different domain or application. The requesting entity of the two parties in this undertaking is called the Relying Party (RP) and the providing entity is known as the Identifying Party (IdP).

[*] The topic I chose to write on is based on the following research paper –

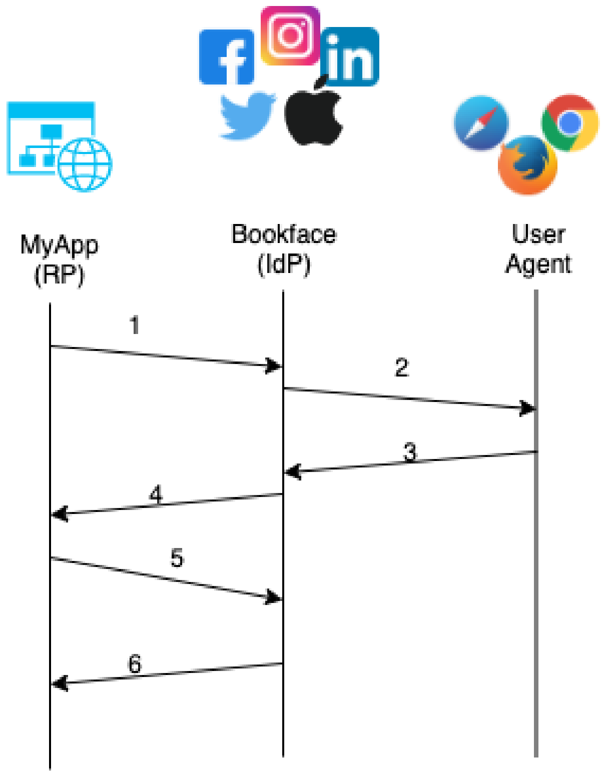
Shernan E., Carter H., Tian D., Traynor P., Butler K. (2015) More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations. In: Almgren M., Gulisano V., Maggi F. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2015. Lecture Notes in Computer Science, vol 9148. Springer, Cham. https://doi.org/10.1007/978-3-319-20550-2_13

Examples of OAuth 2.0 IdP:



OAuth 2.0 has multiple forms of authentication follows for different scenarios – native, web, and mobile applications to suit a wide variety of scenarios. This paper focuses exclusively on the vulnerability that exists within the authorization code flow of the OAuth 2.0.

Diagram 1:



Normal Protocol:

Consider John Doe who has an account on Bookface – also an Identifying Party (IdP). John Doe also uses MyApp. MyApp, the Relying Party (RP), needs access to John's user data provided by Bookface. Steps followed by the protocol are as follows:

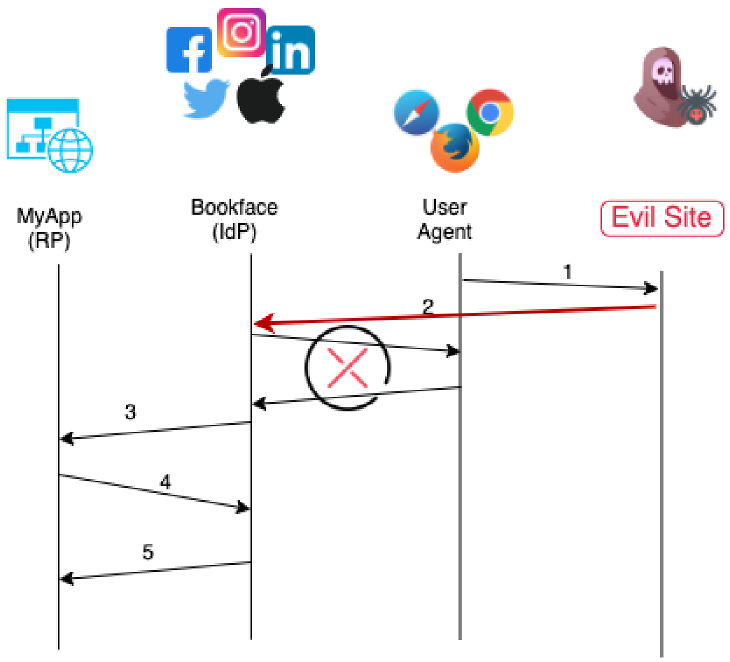
1. MyApp sends an HTTP GET request to Bookface's OAuth 2.0 endpoint containing the following parameters – client_id (the value to determine the service), scope (set of descriptors that describe the data to be accessed), grant_type (authorization grant), and redirect_uri (the URI the user agent or browser to where it will receive the authorization code).

2. Bookface responds to this request by authenticating the user credentials and confirming the consent for access to the data requested by the RP.
3. If confirmed, Bookface sends an authorization code to the redirect_uri.
4. MyApp receives the code at the redirect_uri and makes a POST request to Bookface to exchange this code for final authorization token that can be used to make authorized API calls to Bookface.

Malicious usage of the protocol (CSRF):

This is a form of confused deputy attack where the user's established session is misused by an attacker to make a malicious request as if arising from the authorized user. The attack itself involves an HTTP request on behalf of a victim who logs into the IdP website using stored session data. The malicious URL used in the CSRF attack is embedded within an <img> HTML tag on an innocent looking web page that the user clicks on which leads to a GET request to the URL. If the user has already logged into the IdP website, a session cookie is automatically sent along with the malicious HTTP request. The IdP processes the request as if from a normal user.

Diagram 2:



CSRF Mitigation:

While developers consider this as a low-risk vulnerability, the security community regards this form of attack as a serious threat. An important feature of the above protocol is that while making API calls with the authorization token, additional parameters might be included according to the standard. This is the key feature to mitigate the CSRF attack – that is to include a state parameter along with the API calls made by the RP. The scheme uses a randomly generated token that synchronizes a specific request with a specific user session. Requests are disallowed unless a token is included with every HTTP request and matches the user’s current session token as remembered by the server. The RFC defining OAuth 2.0 provides an entire subsection detailing the attack and necessitating CSRF protection for its authorization endpoint.

Analysis of inadequate implementation:

The paper examines to determine if there’s noncompliant implementation or challenges in implementing the CSRF mitigation correctly. CSRF techniques employed by common RPs (see table below) were examined. A variety of different CSRF prevention techniques were found to exist in practical deployment.

Provider	**Anti-CSRF**	**Provider**	**Anti-CSRF**	**Provider**	**Anti-CSRF**
Battle.net	Forced	Dropbox	Suggested	AOL	No Mention
Github	Forced	Facebook	Suggested	Microsoft	No Mention
LinkedIn	Forced	Google	Suggested	Salesforce.com	No Mention
Reddit	Forced	Instagram	Suggested		
Amazon	Suggested	PayPal	Suggested		

The analysis of these websites was accomplished using an in-house developed OAuth Detector (OAD), a web crawler based on the Beautiful Soup library. OAD crawled top 10k Alexa.com websites to check whether the website makes any OAuth request and if the implementation was correct (by checking if there exists a *state* parameter).

Of all the websites crawled, 302 domains implemented OAuth 2.0 in some form. Of those, 77 implemented OAuth without CSRF protection, a good 25% of the RPs.

Furthermore, the paper goes a step further and investigates four cases to understand the reason why this vulnerability exists at such a scale in live deployments. The websites were chosen based on size of the company, and support for developers. The key points which arise out of this analysis are that there’s –

1. Missing documentation in many of the major IdP implementation support manuals. Sample scripts provided are often implemented by the developers as is to avoid the implementation complexity.
2. Inadequate code samples also lead developers to implement OAuth 2.0 in an insecure manner.
3. Inconsistent Requirements: The IdP fails to provide any tools for correctly implementing the state parameter.
4. Lack of enforcement: In various cases anti-CSRF tokens were not a necessity thus exposing the protocol to vulnerability.

Conclusion:

The paper concludes by recommending that IdP is the actual party responsible for this shortcoming of efficient anti-CSRF implementation across RPs trying to access data from reputable IdPs. The author also argue that the previous research has recommended mitigation techniques to be implemented by RPs.

I agree with the authors of this research that Identifying parties are key responsible entities to help developers implement the anti-CSRF token correctly. This is true due to the fact that although OAuth and CSRF mitigation technique is a standard, different websites have different web access APIs and data structure – therefore it is the responsibility of the IdP to provide accurate guidelines for the developers. The paper has indeed provided a comprehensive foundation to make the technology world aware of the importance of thorough documentation by the providers and correct implementation on the part of the developers.