2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

OAuth-SSO: A Framework to Secure the OAuth-based SSO Service for Packaged Web Applications

Nazmul Hossain
Assistant Professor, Department of Computer Science and engineering
Jessore University of Science & Technology
Jessore-7408, Bangladesh
nazmul.justcse@gmail.com

Md. Alam Hossain
Assistant Professor, Department of Computer Science and engineering
Jessore University of Science & Technology
Jessore-7408, Bangladesh
alamcse_iu@yahoo.com

Md. Zobayer Hossain
Department of Computer Science and engineering
Jessore University of Science & Technology
Jessore-7408, Bangladesh
zobayer130127@gmail.com

Md. Hasan Imam Sohag
Department of Computer Science and engineering
Jessore University of Science & Technology
Jessore-7408, Bangladesh
hasan.i.sohag@gmail.com

* Shawon Rahman, Ph.D.
Associate Professor, Dept. of Computer Science, University of Hawaii-Hilo
Hilo, Hawaii 96720, USA
sRahman@Hawaii.edu
* Corresponding Author

***Abstract*— The OAuth 2.0 is an authorization protocol gives authorization on the Web. Popular social networks like Facebook, Google and Twitter make their APIs based on the OAuth protocol to increase user experience of SSO and social sharing. It is an open standard for authorization and gives a process for third-party applications to obtain users' resources on the resource servers without sharing their login credentials. Single sign-on (SSO) is an identification method that makes allowance for websites to use other, rely on sites to confirm users. OAuth 2.0 is broadly used in Single Sign-On (SSO) service because of its simple implementation and coherence with a diversity of the third-party applications. It has been proved secure in different formal methods, but some vulnerabilities are revealed in practice. In this paper, we mention a general approach to improve the security of OAuth based SSO service for packaged web app. This paper proposes a modified method to execute OAuth flow from such applications with the help of Single sign-on (SSO) manages the life cycle of these applications.**

Keywords—OAuth 2.0 Security; SSO; Packaged Web Apps; Authentication; Authorization

I. INTRODUCTION

Nowadays the development of the Internet, particularly the wide use of web 2.0, web applications (like online shopping, instant messaging and wiki) have to be an important part of our regular lives. The client web app developers, want to utilize this medium to access various user resources, such as photos or videos maintained in some external resource server, in order to provide a richer and connected experience through their apps. To improve the situation, Single Sign-On (SSO) service comes out. In SSO, users are authorized to access various Relying Parties (RP for short) while logging into Identity Provider (IP for short) only at one time, which is openly convenient for users.

In this paper, we propose a general approach to improve the security of OAuth-based SSO service. At first we discuss the parameters and the types of flows defined in the protocol, and design 5 attacks (A1-A5) in some prolepsis. After then the approach containing 9 detection terms (I1-I9) and an app id is provided to check the prolepsis. In last section we arrange an experiment on some typical IPs and RPs. The results show that the approach is simple and easy-to-use.

II. OAUTH 2.0 AUTHENTICATION FLOWS

The OAuth 2.0[1] specification which defines two flows for RPs to gain access tokens: server-flow intended for web applications that receive access tokens from their server-side program logic; and client-flow for JavaScript applications running in web browser [7]. Fig. 1 illustrates the following steps, which exhibit how server-flow works:

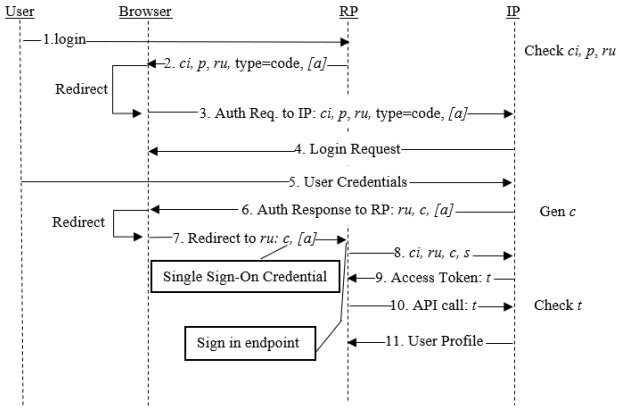

Fig. 1. The Server Flow in OAuth 2.0

1. User X clicks on the social login button, and the browser Y sends this login HTTP request to RP.
2. RP sends response type=code, client ID *ci* (a random unique RP identifier assigned during registration with the IP), requested permission scope *p*, and a redirect URL *ru* to IP via Y to obtain an authorization response.

2324-9013/18/31.00 ©2018 IEEE
DOI 10.1109/TrustCom/BigDataSE.2018.00227

3. Y sends response_type=code, *ci*, p, *ru* and optional a to IP. IP checks *ci*, *p* and *ru* against its own local storage.
4. IP presents a login form to authenticate the user. This step could be omitted if X has already authenticated in the same browser session.
5. X provides her credentials to authenticate with IP, and then consents to the release of her profile information.
6. IP generates an authorization code c, and then redirects Y to *ru* with *c* and *a* (if presented) appended as parameters.
7. Y sends *c* and a to *ru* on RP.
8. RP sends *ci*, *ru*, *c* and *a* client secret s (established during registration with the IP) to IP's token exchange endpoint through a direct communication.
9. IP checks *ci*, *ru*, *c* and *s*, returns an access token t to RP.
10. RP makes a web API call to IP with *t*.
11. IP validates t and returns X's profile attributes for RP to create an authenticated session.

The client-flow is designed for applications that cannot embed a secret key, such as JavaScript clients [3]. The access token is returned directly in the redirect URI, and its security is handled in two ways: (1) The IP validates whether the redirect URI matches a pre-registered URL to ensure the access token is not sent to unauthorized parties; (2) the token itself is appended as an URI fragment (#) of the redirect URI so that the browser will never send it to the server, and hence preventing the token from being exposed in the network. Fig.2 illustrates how client-flow works:

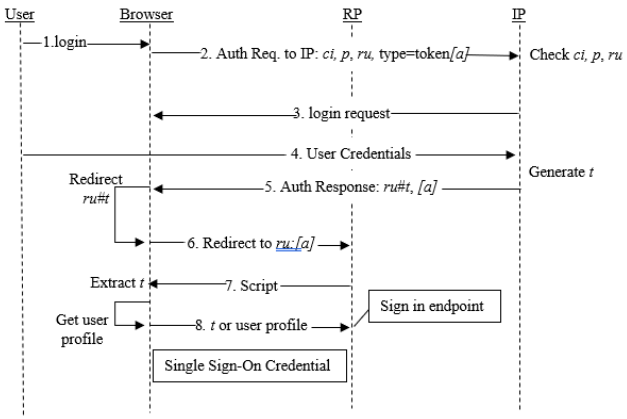

Fig. 2. The Clint Flow in OAuth 2.0

1. User X initiates an SSO process by clicking on the social login button rendered by RP.
2. B sends response_type=token, client ID *ci*, permission scope p, redirect URL *ru* and an optional state parameter *a* to IP.
3. Same as sever-flow step 4 (i.e., authentication).
4. Same as sever-flow step 5 (i.e., authorization).
5. IP returns an access token *t* appended as an URI fragment of *ru* to RP via Y. State parameter *a* is appended as a query parameter if presented.
6. Y sends a to *ru* on RP. Note that Y retains the URI fragment locally, and does not include *t* in the request to RP.
7. RP returns a web page containing a script to Y. The script extracts *t* contained in the fragment using JavaScript command such as document.location.hash.
8. With t, the script could call IP's web API to retrieve X's profile on the client-side, and then send X's profile to RP's sign-in endpoint.

III. Releted Work

In theory, several formal approaches have been used to examine the OAuth, such as Alloy framework used by Pai et al. [8] and all the results were included in the official OAuth security guide [9]. Thus, the implementation following the guide is secure in theory. On empirical analysis, Wang et al. [10] focused on the original web traffic going by the browser and discovered several logic flaws in some SSO services (e.g. Google ID, Facebook) , which are used by attackers to tamper with the authentication messages.

Actually, all the existing approaches are deficient. The formal analyses were conducted in the abstract models, and some implementation details could be left out, which has been verified by the empirical studies; while the existing empirical studies exposed the vulnerabilities in the case analyses, which could omit some proofs.

IV. Proposed Framework

Our analytic framework is to provide a general approach to secure the OAuth-based SSO service. In the stage of protocol analysis, we review the protocol carefully to reveal the usages, requirements and potential risks of the five variable parameters and session S. Five attacks (identified by A1 to A5) based on the former are provided in the latter stage, all of which have been proved available in the following experiment and the analysis on the presuppositions of the five attacks reveals that we need only to execute a detection on item I1 to I9 as follows to evaluate the security of an SSO service:

I1: Whether HTTPS protection is deployed by RP.

I2: Whether an unpredictable state parameter is used by RP.

I3: Whether RP is prevented against CSRF attack.

I4: Whether RP stores the access token in the cookie or URI.

I5: Whether the code is cross in use.

I6: Whether IP supports the two response types simultaneously and indiscriminately.

I7: Whether any redirection URI in the realm of RP could pass IP's checking.

I8: Whether the token is a bearer token.

I9: Whether a mechanism to end the session S is provided.

V. Analysis On Proposed Framework

A. Protocol Analysis

In the authentication flows, five variable parameters are defined, and two authentication sessions are created. In the rest part of this section, we discuss the usages, requirements and potential risks of the parameters and the session S. [11]

X1. Response Type (response_type)

Usage: It is set by RP and used to select which flow to be used in the following sequence, that is, to decide the way for RP to gain an access token.

Risk: As mentioned before, the both flows are similar in user's view, so attackers could set response_type = token, and gain the access token from the browser through a script without user's awareness.

X2. Redirection URI (redirect_uri)

Usage: It is set by RP and used to decide the destination of request C6 or T6, which receives the code or access token.

Risk: The receiver, besides an attacker, of the code or access token could log into RP as the victim, and gain the victim's profile.

X3. State Parameter (state)

Usage: It is generated by RP for tracing the session between browser and RP to prevent against CSRF attack.

Risk: It is an optional parameter. If it is lost, another countermeasure against CSRF attack MUST be taken specified in the protocol.

X4. Authorization Code (code)

Usage: It is sent to RP in the server flow and used for RP to request an access token from IP.

Risk: If gaining it and using it before the victim, the attacker could impersonate the victim and log into RP.

X5. Access Token (access_token)

Usage: It is generated by IP and used for RP to make web API calls to gain or handle user's profile. RP use the received profile to authenticate the user.

Risk: Due to the decryptographic design of OAuth, the access token is often implemented as a bearer token. That is to say, any bearer of the token, e.g. an attacker, could access or handle user's profile.

S. Session with IP

Usage: It is created by IP in step C3 or T3 and used to authenticate the user for the latter login.

Risk: After created, the session keeps alive and is only bound to cookies invisible to the user, so other mechanisms should be employed to clear the session.

In conclusion, the misuse of each of X1 to X5 and S could result in unsecure factors, thus understanding the strict requirements are a precondition for implementing an invulnerable OAuth-based SSO service.

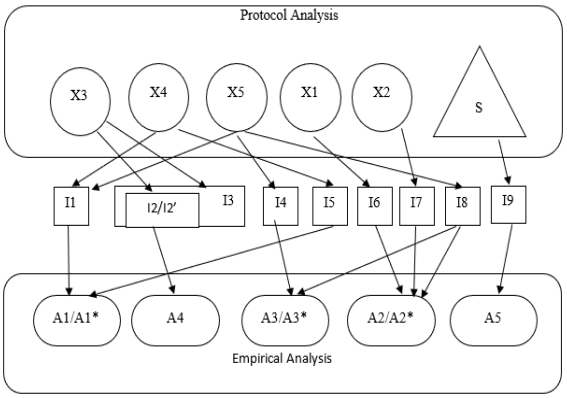

Fig. 3. Analytic Framework of Detecting the Security on Parameters and Session S [15].

B. Empirical Analysis

In this section, accordingly all the necessities, 10 IPs and 136 authentications used by 50 RPs are experimented, and the outcome denote that risks lay in the cautiousness although the OAuth-based SSO service is broadly used. [11]

A1. Theft & Embezzling Authorization Code

In server flow, HTTPS protection is used for all the contacts with IP, but not for one or more submitting the authorization code to 90% of RPs. Cross use of the code is not Identified in our study, but it is mentioned that the authorization code created by weibo.com for RP1 can be used to log into RP2 . In this situation, the risk is of larger damage.

Strikingly, 30% of RPs defend their own login page with HTTPS, but only 9.56% of RPs provide HTTPS protection for the authorization code.

A2. Stealing Access Token via XSS

All the detected IPs sustain the two streams and try not to limit's their RPs to choose that flow to utilize. Thusly, XSS vulnerability in redirect_uri page on RPs utilizing the user flow, as Sun et al. [6] put it, permits attackers to dispatch the client-flow sequence & to correct the access token with the script.

A3. Eavesdropping or Stealing Access Token

The bearer token is deployed in all the detected IPs. These is the only identity for RP to access clint's profile. So RP will undoubtedly ensure its confidentiality.

A4. CSRF Attack

The results explicit that 44.12% of RPs take CSRF attack into consideration, and 39.71% adjust the suggestion utilizing the state parameter. It is important that a little part of RPs set the state parameter as a permanent value, and that about 25% of RPs try not to check the parameter exactly & accept the request except the parameter, all of which fall flat in opposition to CSRF attack.

A5. Risk of the Implicit Session with IP

As the session lays in the browser implicitly, the user can't end it by closing some page after login, which may initiate implicit session with IP. For this reason, logging out of the IP is a recommendation for a complete solution to refrain from the aforementioned risk. Nevertheless, none of the detected RPs maintains a mechanism to end session S though parts of IPs offer an interface to log out.

VI. Result

To begin an assessment process, the evaluator signs into the RP in question using both traditional and SSO options through a Firefox browser. The browser is augmented with an add-on we designed that records and analyzes the HTTP requests and responses passing through the browser. To resemble a real-world attack scenario, we implemented a website, denoted as attacker.com, which retrieves the analysis results from the trace logs, and feeds them into each assessment module described below. Table 1 shows the summary of our evaluation results. We found 42% of RPs use server flow, and 58% support client-flow; but all client-flow RPs use Facebook SDK instead of handling the OAuth protocol themselves.

Table 1: The Percentage of RPs that is vulnerable to each exploit [12]

RPs			SSL (%)		Vulnerabilities (%)				
Flow	**N**	**%**	**T**	**S**	**A1**	**A2**	**A3**	**A4**	**A5**
Client	56	58	20	6	25	55	43	16	18
Server	40	42	29	15	7	36	21	18	20
Total	96	100	49	21	32	91	64	34	38

VII. Conclution

OAuth 2.0 is attractive to RPs and easy for RP developers to implement, but our investigation suggests that I is too simple to be secure completely. Unlike conventional security protocols [13], OAuth 2.0 is designed without sound cryptographic protection [14], such as encryption, digital signature, and random nonce [15]. Compared to server flow, client-flow is inherently insecure for SSO. Based on these insights, we believe that OAuth 2.0 at the hand of most developers without a deep understanding of web security [2] is likely to produce insecure implementations.

To protect web users in the present form of OAuth SSO systems, we suggest simple and practical mitigation mechanisms. It is important for current IPs and RPs to adopt those protection mechanisms in order to prevent large-scale security breaches that could compromise millions of web users' accounts on their websites. In particular, the design of server flow makes it more secure than client-flow, and should be adopted as a preferable option, and IPs should offer explicit flow registration and enforce single-use of authorization code. Furthermore, JavaScript SDKs play a crucial role in the security of OAuth SSO systems; a thorough and rigorous security examination of those libraries is an important topic for future research.

References

[1] The OAuth 2.0 authorization framework, IETF Std. RFC6749, 2012

[2] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In Proceedings of IEEE Symposium on Security and Privacy, 2010.

[3] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Fast and precise in-browser JavaScript malware detection. In Proceedings of the 20th USENIX Conference on Security, Berkeley, CA, USA, 2011.

[4] A. Barth, J. Caballero, and D. Song. Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In Proceedings of the 30th IEEE Symposium on Security and Privacy, SP '09, pages 360{371, Washington, DC, USA, 2009.

[5] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08), pages 75{88, New York, NY, USA, 2008. ACM

[6] S.-T. Sun, K. Hawkey, and K. Beznosov. Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. Computers & Security, 2012.

[7] W. Robertson and G. Vigna. Static enforcement of web application integrity through strong typing. In Proceedings of the 18th Conference on USENIX Security Symposium, 2009.

[8] S. Pai, Y. Sharma, S. Kumar, R.M. Pai, and S. Singh, "Formal verication of OAuth 2.0 using Alloy framework," in Proc. CSNT'11, 2011, p. 655-659.

[9] OAuth 2.0 Threat Model and Security Considerations, IETF Std. RFC6819, 2013.

[10] Wang, S. Chen, and X. Wang, "Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single sign-on web services," in Proc. SP'12, 2012, p. 365-379.

[11] R Zhu, J Xiang, D Zha. Research on the Security of OAuth-Based Single Sign-On Service. The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014.

[12] San-Tsai Sun and Konstantin Beznosov. The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems. CCS '12 Proceedings of the 2012 ACM conference on Computer and communications security, Pages 378-390, Raleigh, North Carolina, USA — October 16 - 18, 2012.

[13] V Beltran. Characterization of web single sign-on protocols. IEEE Communications Magazine, 15 July 2016.

[14] Daniel Fett, Ralf Küsters, Guido Schmitz. A Comprehensive Formal Security Analysis of OAuth 2.0. CCS '16 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security Pages 1204-1215, Vienna, Austria — October 24 - 28, 2016

[15] V Sucasas, G Mantas, A Radwan, J Rodriguez. An OAuth2-based protocol with strong user privacy preservation for smart city mobile e-Health apps. Communications (ICC), 2016 IEEE International Conference on Communications (ICC), 22-27 May 2016.