Journal of Korea Multimedia Society Vol. 19, No. 2, February 2016(pp. 308-315)
http://dx.doi.org/10.9717/kmms.2016.19.2.308

A Secure Social Networking Site based on OAuth Implementation

Otieno Mark Brian†, Kyung-Hyune Rhee††

ABSTRACT

With the advancement in the area of cloud storage services as well as a tremendous growth of social networking sites, permission for one web service to act on the behalf of another has become increasingly vital as social Internet services such as blogs, photo sharing, and social networks. With this increased cross-site media sharing, there is a upscale of security implications and hence the need to formulate security protocols and considerations. Recently, OAuth, a new protocol for establishing identity management standards across services, is provided as an alternative way to share the user names and passwords, and expose personal information to attacks against on-line data and identities. Moreover, OwnCloud provides an enterprise file synchronizing and sharing that is hosted on user's data center, on user's servers, using user's storage. We propose a secure Social Networking Site (SSN) access based on OAuth implementation by combining two novel concepts of OAuth and OwnCloud. Security analysis and performance evaluation are given to validate the proposed scheme.

Key words: OAuth, OwnCloud, Social Networking Sites

1. INTRODUCTION

1.1 Background

Internet-based social networking sites have created a revolution in social connectivity. Social networking sites are Internet-based services that allow people to communicate and share information with a group. Facebook, Twitter, Google+, LinkedIn and other social networks have become an integral part of online lives. Social networks are a great way to stay connected with others through the sharing of information such as photos, videos, and personal messages [1]. In the traditional client-server authentication model according to D. Hardt [2], the client requests a protected resource on the server by authenticating with the server using the resource owner's credentials. In order to provide third-party applications access to restricted resources, the resource owner shares its credentials with the third party.

This has in turn created numerous limitations:

- Third-party applications have a broad access to the resource owner's protected resources, leaving resource owners without any ability to restrict duration of access and without the ability to limit subset of resources.
- Resource owners cannot revoke access to an individual third party without revoking access to all third parties.
- Third-party applications are required to store the resource owner's credentials for future use.
- Compromise of any third-party application

※ Corresponding Author : Kyung-Hyune Rhee, Address: (48513) 45, Yongso-ro, Nam-Gu. Busan, Republic of Korea, TEL : +82-51-629-6247, FAX : +82-51-626-4887, E-mail : khrhee@pknu.ac.kr
Receipt date : Jan. 22, 2016, Approval date : Feb. 5, 2016

† Dept. of IT Convergence and Application Engineering, Pukyong National University
(E-mail : mbotieno@gmail.com)
†† Dept. of IT Convergence and Application Engineering, Pukyong National University
※ This work was supported by a Research Grant of Pukyong National University (2015 year).

results in compromise of the end-user's password and all of the data protected by that password.

- There are great weaknesses that are inherent in passwords, yet servers are required to support password authentication.

These issues above need to be addressed, OAuth 2.0 therefore introduces an authorization layer and separates the roles of the client from that of the resource owner. The client requests access to resources that are controlled by the resource owner and hosted by the resource server, and is then issued a varied set of credentials other than those of the resource owner. The client obtains an access token which is a string denoting a specific scope, lifetime, and other access attributes, instead of using the resource owner's credentials to access protected resources. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server [3].

1.2 Overview and Contribution

OAuth 2.0 support is a key requirement which will make OwnCloud attractive as a platform for third-party developers who need to integrate OwnCloud into their applications. OAuth is a security protocol that allows third-party applications to request access to protected information without providing the username and the password to other party. Currently, third-party applications have to use Basic Authentication which involves sending the username and password to access the OwnCloud instance which has the following disadvantages:

- Users have to provide their credentials to third-party applications. If one of the third-party providers has been compromised then OwnCloud login will be lost.
- An authentication can only be revoked by changing the user password which is suboptimal.
- Third-party developers do currently have access to the whole OwnCloud instance; they even could change your password.

Instead of using passwords for authorization, OAuth is using unique tokens for every client. In OAuth, the client requests access to the needed resources (scopes) by redirecting the user to a website where he has to approve these permissions.

In this paper, we propose a secure sharing platform between OwnCloud cloud storage and internet based social networking sites. This resolves the limitation of the current OwnCloud infrastructure and third-party applications by providing a security protocol that allows third-party applications to request access to the protected information without providing the username and the password to other party. The proposed protocol entails integrating third-party SSO login into the OwnCloud framework, thereby allowing third party internet social networking sites to have access to all the media files stored within the cloud storage. In order to achieve these goals, we propose the integration of OAuth 2.0 security protocol into the OwnCloud platform. OAuth 2.0 will ensure security by ensuring the proper authorization flow is followed during communication between OwnCloud and the social networking sites. In addition, it performs authentication of the resource owner/clients through the Authorization Server. Moreover, it ensures a token is issued that not only authorizes but also dictates the scope and lifetime of a given action.

This implementation will be of benefit to different users, developers and administrators on different levels. Users will be able to grant third-party applications access to their data without providing their passwords or granting access to the whole instance. Users and administrators will be able to see which applications have access to which data and manage them. And lastly, developers will be able to use standard libraries to integrate with OwnCloud.

2. PRELIMINARY AND RELATED WORK

2.1 Preliminary

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. The OAuth 2.0 authorization protocol standardizes delegated authorization on the Web. Popular social networks such as Facebook, Google and Twitter implement their APIs based on the OAuth protocol to enhance user experience of social sign-on and social sharing. In [4], F. Yang et al. describes OAuth (open standard for authorization) as a protocol that provides a generic framework to let a resource owner authorize third-party to access the owner's resource held at a server without revealing to the third-party the owner's credentials (such as user-name and password) [2], [5].

The OAuth 2.0 security protocol defines four roles which helps it accomplish the purpose it is tasked with reliably [6].

- **Resource owner** - This refers to an entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
- **Resource server** - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- **Client** - An application making protected resource requests on behalf of the resource owner and with its authorization.
- **Authorization server** - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

OAuth 2.0 flow describes the interaction between the four roles and includes the following steps:

1. The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner, or preferably indirectly via the authorization server as an intermediary.
2. The client receives an authorization grant, which is a credential representing the resource owner's authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the authorization server.
3. The client requests an access token by authenticating with the authorization server and presenting the authorization grant.
4. The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.
5. The client requests the protected resource from the resource server and authenticates by presenting the access token.
6. The resource server validates the access token, and if valid, serves the request.

2.2 Related Work

There has been several implementations and integrations of secure an extensive sharing to internet social networking sites. These implementations normally involve the integration of the capability to login into host sites via a popular social networking site account.

Dropbox and OwnCloud are the two most adopted cloud storage platforms. OwnCloud does not however support the ability to directly share the media stored in it to the social networking sites. The integration involves allowing the capability to login into OwnCloud via social networking sites such as Facebook. OwnCloud have not embraced this feature because of the implications that arise from such integration.

OwnCloud currently uses the Lightweight Directory Access Protocol (LDAP) and the Web Distributed Authoring and Versioning (WebDAV). LDAP is a directory service protocol that runs on a layer above the TCP/IP stack. It provides a mechanism used to connect to, search, and modify Internet directories. The LDAP directory service is based on a client-server model. Web Distributed Authoring and Versioning (WebDAV) is an extension of the Hypertext Transfer Protocol (HTTP) that allows clients to perform remote Web content authoring operations. The WebDAV protocol provides a framework for users to create, change and move documents on a server, typically a web server or web share. LDAP on one hand is used for user authentication while WebDAV on the other hand is used for file access. [7]

Other identity management protocols have been used before. SAML(Security Assertion Markup Language) is a set of standards that have been defined to share information about who a user is, what his set of attributes are, and give you a way to grant or deny access to something or even request authentication. SAML is an XML-based open standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. SAML deals with XML as the data construct or token format. OAuth's tokens on the other hand can be binary or JSON. SAML has bindings that use HTTP such as HTTP POST binding, HTTP REDIRECT binding, but it is of note to point out that there is no restriction on the transport format. SOAP or JMS or any other transport can be used to send SAML tokens or messages. However, OAuth uses HTTP exclusively. Even though SAML was designed to be applicable openly, it is typically used in Enterprise SSO scenarios; within an enterprise or enterprise to partner or enterprise to cloud scenarios. On the contrary, OAuth has been designed for use with applications on the internet, primarily for delegated authorization of internet resources. OAuth is designed for Internet Scale. The problem with this SAML example is its only one specialized use case out of many [8].

3. PROPOSED PROTOCOL

3.1 System Model

The proposed framework involves implementing the OAuth 2.0 security protocol into the OwnCloud platform. The OAuth 2.0 server authentication flow is initiated when an OwnCloud account uses the integration for the first time. The OwnCloud user must log in to their account and give permission to the third-party social networking sites application so as to be able to access the OwnCloud account. The server authentication flow consists these main processes:

- An authorization request is made by the third-party application
- An authorization code response is issued by the authorization server
- Using the authorization code, the third-party application makes an access token request. An access token is issued by the authorization server as a response.

The communication between the user, the third-party application, the OwnCloud, and the OAuth 2.0 Authorization Server is summarized by four main roles. In the proposed system model, these roles are categorized as follows. Firstly, the Resource Server is represented by OwnCloud application. The User represents the Owner/User while the Authorization Server is represented by the OAuth 2.0 protocol. Lastly, the Third-Party Application represents the Social Networking Sites such as Facebook and Twitter.

The proposed process communication flow is as depicted in the Fig. 1. Below;

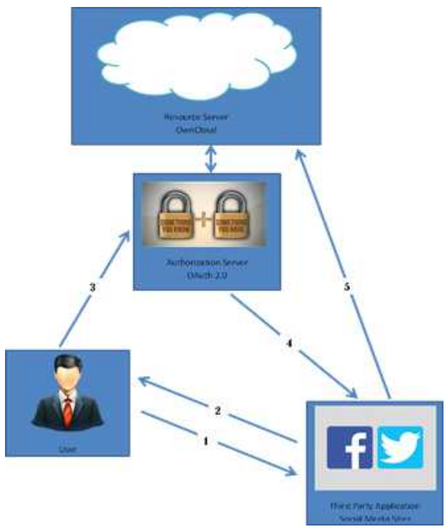

Fig. 1. Proposed system model.

1. The User visits the Third-Party Application through the Social Media Sites webpage

2. The application directs the user to the OAuth 2.0 authorization server security protocol

3. The User is authenticated with OAuth 2.0 and then grants the Third-party Application access to their account.

4. The Authorization Server redirects the User to the application using the redirect URI. It also provides an authorization code if the user granted access to the application. The User is redirected to the application's redirection endpoint, the redirect_url. This is done with an authorization code in the code URL parameter.

5. The application then exchanges the authorization code for an access token for use in all API calls for that account. The application uses the Authentication Code to obtain Access and Refresh Tokens using a POST request to the token endpoint. The authorization server validates the authorization code and, if valid, a JSON response containing the access token, refresh token, access token expiration time, and token type is issued.

3.2 Token Assertion

According to the OAuth 2.0 specification, RFC 6749 [2], it very specifically punts on this issue in section 7: "The methods used by the resource server to validate the access token (as well as any error responses) are beyond the scope of this specification but generally involve an interaction or coordination between the resource server and the authorization server." This results in a limitation in the access token validation. In this proposed model, the resource servers assert the token issued by the authorization server.

There are two ways to avert these limitations and these solutions are applied according to the scenarios present. For small deployments, it can look it up in a database. In many instances, the RS and the AS are usually co-located and very tightly bound so they have access to the same data store. When the AS part of the server mints a token, it drops the token or a hash of it into a database along with all of the information about the token that will be needed to make an authorization decision. When that token comes back in later, the RS part of the server just needs to look up the token value or its hash and pluck any other bits of data that is needed from that record in order to authorize or deny the request being made.

Secondly, in an instance where there are multiple RS's and a single AS, then there is need for a means to communicate all that meta-information surrounding the token including what scopes it has, who authorized it, what client it was authorized for, when it expires from the AS to the RS. The use of a structured token value like JSON Web Token (JWT) is recommended. JWT is a compact, URL-safe means of representing claims to be transferred between two parties [9].

JWTs are good constructs being that it's a blob of JSON that can be signed and encrypted in a way that won't get mucked up in transit. JWTs define a set of common claims, such as issuer, audience, subject, and other bits needed for a security object. The RS gets handed a JWT, it parses the JWT, checks the signature or decrypts it, reads the claims, sees who the token is for and what it is for and if it's expiry. The RS therefore gets every-thing it needs from that.

4. IMPLEMENTATION AND ANALYSIS

4.1 Implementation

Access tokens can be used as proof of authenti-cation. Since an authentication usually occurs ahead of the issuance of an access token, it is pos-sible to consider reception of an access token of any type proof that such an authentication has occurred.

Access of a protected API can also be used as a proof of authentication. Since the access token can be traded for a set of user attributes, it is viable to assume that possession of a valid access token is enough to prove that a user is authenticated. This is especially true in cases where the token was freshly minted in the context of a user being authenticated at the authorization server.

Authentication of the user who is trying to ac-cess OwnCloud via a third party is also established based on the integration of the OAuth 2.0 protocol which acts to establish identity management. A dialogue notification pops up upon any access giv-ing the owner (OwnCloud) the authority to either grant access or deny access and also to determine the scope and sharing limitations of the access.

Token Assertion is done by the by Resource Server. This works on the assumption that the RS will be able to call the AS for each token that it sees, and that there is no problem with the extra network traffic. Thus there is the accuracy/per-formance tradeoff in most networked systems: you can have live information by calling the author-itative source in real time (using introspection) or you can have self-contained information that you don't have to make a network call for (using JWT). You can, of course, cache the introspection call, and most implementations do this on the client side, at least to an extent.

4.2 Analysis

4.2.1 Performance evaluation

To evaluate the performance of the server after applying before and after integration of the OAuth 2.0 security protocol, a load tester JMeter is used. The Apache JMeter is an open source java soft-ware application designed to load test functional behavior and measure performance. This works by simulating multiple users to have access to the server concurrently. There are three main parame-ters that are considered in this simulation.

- Number of threads: This represents the num-ber of users connected to the target website.
- Ramp-Up period: This denotes the time it takes for a user to start a new session.
- Loop count: This denotes the number of re-quest for each user.

The number of users is then varied in sets of tens from 10 all the way to 100 for this particular scenario. The application is set to handle 10 user requests with a Ramp-Up period of 1 second. The Rump-Up period determines how long the next user should take to begin a new session. The figure 2 below shows a comparison of the performance results between three different identity manage-ment protocols OAuth 2.0, SAML and OpenID.

The result shows that the performance of OwnCloud as the number of users' increases con-siderably increases the response time.

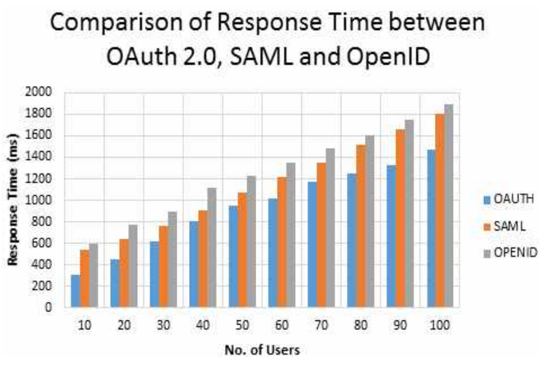

Fig. 2. Comparison between performance test results.

Using the same values in the required parameters, it was observed that the response time according to the increase of the users was higher in an implementation instance where OpenID protocol was integrated as compared to an instance where either OAuth 2.0 or SAML protocol was used. This means that the integration of OAuth security protocol with the OwnCloud environment results in a better performance compared to other identity management protocols.

JWTs define a set of common claims which if packed into a token could raise some issues:

- The token could become rather large and unwieldy if there is a lot to say about the authorization context.
- There is the risk of the client being able to read what is in the token, which might leak sensitive information.

4.2.2 Security Analysis

Access token can be used as a proof of authentication. Since an authentication usually occurs ahead of the issuance of an access token, it can be considered the reception of an access token as any type of proof of authentication. OAuth tokens are opaque to clients, which mean they do not have to read the token to use it, but that does not mean that a client cannot try to read the token and get something useful out of it. This can be combatted by encrypting the token, but even the JWT specification says that the best way to avoid privacy leakage issues is to just not put sensitive information inside the token itself. And it also assumes that the owner is okay with tokens being good until they expire, because if the RS is parsing the token on its own, there is no good way to revoke a token once it is in flight.

However, this can be combatted by having short enough timeouts on the tokens. Alternatively, the RS can have a service it calls at runtime to get information about the token in the context of its authorization decision, then find out in real time if the token has been revoked or not. And if it is making that call, it could also just as easily find out all of the important meta-information about that token. Token Introspection defines a very simple HTTP service that lets an RS send the token over in a POST and get back a JSON document that says what the token is good for. Introspection re-uses the claims defined in JWT and adds a few of its own. The RS authenticates to the AS during this call so that nobody can go search for token information.

5. CONCLUSION

In this paper, we proposed the implementation of OAuth 2.0 security protocol into the OwnCloud platform environment. This implementation is to enable secure access of the OwnCloud stored data during communication between OwnCloud and other third-party applications. The third-party applications considered in this particular case are the online social networking sites. We have used Facebook and Twitter as the demonstration social networks in order to implement this particular protocol. The implementation has thus proved that the integration has provided secure access by allowing OAuth 2.0 security protocol to act as the identity management tool between OwnCloud and the Social Networking Sites by providing user verification and authentication. OwnCloud is therefore tasked with the authority to grant or deny access to any third-parties and to also determine the scope of what can be accessed or shared. It has also further prevented the conventional method of sharing user credentials which include username and password.

The performance result testing for OwnCloud with OAuth 2.0 security protocol integration was plotted in a comparison graph with the use of the same values in the required parameters. Based on the performance results, it was observe that the response time with an increase in the number of users was higher in an OwnCloud instance where OpenID and SAML protocols were used as opposed to an instance where OAuth protocol was integrated. This means that the integration of OAuth security protocol within the OwnCloud environment has resulted in a better performance.

REFERENCES

[1] Tae-Wong Seo, Man-Gon Park and Chang-Soo Kim, “Design and Implementation of the Extraction Mashup for Reported Disaster Information on SNSs,“ *Journal of Korea Multimedia Society,* Vol. 16, No. 11, pp. 1297-1304, 2013.

[2] D. Hardt, *The OAuth 2.0 Authorization Framework,* RFC 6749, 2012.

[3] A. Santana de Oliveira, G. Serme, and Y. Lehmann, "Platform-level Support for Authorization in Cloud Service with OAuth 2," *Proceedings of Intercloud Workshop Co-located with IEEE International Conference on Cloud Engineering,* pp.458-465, 2014.

[4] Yang and S. Manoharan, "A Security Analysis of the OAuth Protocol," *Proceeding of IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing,* pp. 271-276, 2013.

[5] Hammer-Lahav, *The OAuth 1.0 protocol,* The Internet Eng. Task Force RFC 5849, 2010.

[6] Er. Gurleen Kaur and Er. Deepak Aggarwal, “A Survey Paper on Social Sign-On Protocol OAuth 2.0,” *Journal of Engineering Computers & Applied Sciences,* Vol. 2, No. 6, pp. 93-96, 2013.

[7] ownCloud’s Architecture Overview, https://owncloud.com/whitepapers (accessed, 11, Feb., 2016).

[8] Paulo Jorge Correia, *Providing Single Sign on (SSO) with Enterprise Identity Services and Directory Integration,* Cisco Public BRKUCC-2664, USA, 2015.

[9] JSON Web Token (JWT), https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32 (accessed, 11, Feb., 2016).

Mark Brian Otieno

Aug. 2011. Daystar University in Kenya (B. Sc.)
Feb. 2016. Department of IT Convergence and Application Engineering in Pukyong National University (M.Sc,)

Field of Study : Mobile Security, Access Cpontrol, User Authentication

Kyung-Hyune Rhee

Feb. 1982. Department of Mathematical Education, Kyungpuk National University. (B. Sc)
Feb. 1985 Department of Applied Mathematics in Korea Advanced Institute of Science and Technology (M. Sc.)
Aug. 1992 Department of Mathematics in Korea Advanced Institute of Science and Technology (Phd..)
Mar. 1993～Present Professor at Department of IT Convergence and Application Engineering in Pukyong National University

Field of Study : Intelligence Security, Cryptographic Protocols, Applied Cryptography, Multimedia Security, IoT security