A Survey Paper on Social Sign-On Protocol OAuth 2.0

Er. Gurleen Kaur, Research Scholar, BBSBEC Fatehgarh Sahib(Punjab)
Er. Deepak Aggarwal, Professor, BBSBEC, Fatehgarh Sahib (Punjab)

ABSTRACT
This paper presents the amalgamation of different social sites and the concept of social sign-on. Latest Trends of using Facebook Platform, Google Friend Connect, Twitter etc have elevated the concept of social sign-on. These social-networks connect services increase access to and enrich user data in the Social Web, although they also present several security and privacy challenges. It 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. In this paper, we have presented the basic concept of the social sign-on protocol OAuth 2.0 and few challenges faced by this protocol.

Keywords: Social sign-on, OAuth 2.0, third party, Resource server, token

Introduction
Social-networking websites let users build social connections with family, friends, and coworkers. Users can also build profiles for storing and sharing various types of content with others, including photos, videos, and messages. Updating user profiles with interesting content is a form of self-expression that increases interaction in such sites. To encourage this interaction and provide richer content, social-networking sites expose their networks to Web services in the form of online application programming interfaces. These APIs allow third-party developers to interface with the social-networking site, access information and media posted with user profiles, and build social applications that aggregate, process, and create content based on users’ interests. OAuth 2.0 is one such application that helps to overcome the issues of traditional client-server authentication model.

Working Of OAUTH 2.0
In the traditional client-server authentication model, the client requests an access restricted resource (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 creates several problems and limitations:
1. Third-party applications are required to store the resource owner's credentials for future use, typically a password in clear-text.
2. Servers are required to support password authentication, despite the security weaknesses inherent in passwords.
3. Third-party applications gain overly broad access to the resource owner's protected resources, leaving resource owners without any ability to restrict duration or access to a limited subset of resources.
4. Resource owners cannot revoke access to an individual third-party without revoking access to all third-parties, and must do so by changing their password. Compromise of any third-party application results in compromise of the end user's password and all of the data protected by that password.

OAuth 2.0 (Open Authentication) Protocol addresses these issues by introducing an authorization layer and separating the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner. Instead of using the resource owner's credentials to access protected resources, the client obtains an access token - a string denoting a specific scope, lifetime, and other access attributes. 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.

For example, an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo sharing service (resource server), without sharing her username and password with the printing service. Instead, she authenticates directly with a server trusted by the photo sharing service (authorization server) which issues the printing service delegation-specific credentials (access token).
1. Roles in OAuth 2.0
OAuth defines four roles:

1. **Resource owner**
   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.

2. **Resource server**
   The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

3. **Client**
   An application making protected resource requests on behalf of the resource owner and with its authorization.

4. **Authorization server**
   The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

2. Protocol Flow

![Abstract Protocol Flow](image)

Fig. 1 Abstract Protocol Flow

The abstract flow illustrated in Fig. 1 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 (as shown), 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.

3. Authorization Grant
An authorization grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token.

**There are four grant types:**

1. Authorization code,
2. Implicit,
3. Resource owner password credentials, and

**Authorization Code**
The authorization code is obtained by using an authorization server as an intermediary between the client and resource owner. Instead of requesting authorization directly from the resource owner, the client directs the resource owner to an authorization server, which in turn directs the resource owner back to the client with the authorization code. Before directing the resource owner back to the client with the authorization code, the authorization server authenticates the resource owner and obtains authorization. Because the resource owner only authenticates with the authorization server, the resource owner's credentials are never shared with the client.

**Implicit**
The implicit grant is a simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. In the implicit flow, instead of issuing the client an authorization code, the client is issued an access token directly (as the result of the resource owner authorization). The grant type is implicit as no intermediate credentials (such as an authorization code) are issued (and later used to obtain an access token).
Resource Owner Password Credentials
The resource owner password credentials (i.e. username and password) can be used directly as an authorization grant to obtain an access token. The credentials should only be used when there is a high degree of trust between the resource owner and the client (e.g. the client is part of the device operating system or a highly privileged application), and when other authorization grant types are not available (such as an authorization code).

Client Credentials
The client credentials (or other forms of client authentication) can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously arranged with the authorization server. Client credentials are used as an authorization grant typically when the client is acting on its own behalf (the client is also the resource owner), or is requesting access to protected resources based on an authorization previously arranged with the authorization server.

4. Access Token
Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server. The token may denote an identifier used to retrieve the authorization information, or self contain the authorization information in a verifiable manner (i.e. a token string consisting of some data and a signature). The access token provides an abstraction layer, replacing different authorization constructs (e.g. username and password) with a single token understood by the resource server. This abstraction enables issuing access tokens more restrictive than the authorization grant used to obtain them, as well as removing the resource server's need to understand a wide range of authentication methods. Access tokens can have different formats, structures, and methods of utilization (e.g. cryptographic properties) based on the resource server security requirements.

5. Refresh Token
Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner). Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token (i.e. step (D) in Fig. 1). A refresh token is a string representing the authorization granted to the client by the resource owner. The string is usually opaque to the client. The token denotes an identifier used to retrieve the authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers.

Challenges Faced By Protocol
However, with the emergence of new OAuth Protocol 2.0, simplicity and performance increased, but also many limitations of OAuth 2.0 came into existence:

1. CSRF Attack
OAuth 2.0 is prone to CSRF (cross-site request forgery) attack. It's also known as session riding or XSRF.

2. Unbounded tokens
In 1.0, the client has to present two sets of credentials on each protected resource request, the token credentials and the client credentials. In 2.0, the client credentials are no longer used. This means that tokens are no longer bound to any particular client type or instance.

3. Bearer tokens
2.0 got rid of all signatures and cryptography at the protocol level. Instead it relies solely on TLS (Transport Layer Security).

4. Expiring tokens
2.0 tokens can expire and must be refreshed. This is the most significant change for client developers from 1.0 as they now need to implement token state management. The reason for token expiration is to accommodate self-encoded tokens – encrypted tokens which can be authenticated by the server without a database look-up. Because such tokens are self-encoded, they cannot be revoked and therefore must be short-lived to reduce their exposure. Whatever is gained from the removal of the signature is lost twice in the introduction of the token state management requirement.
5. **Grant types**
In 2.0, authorization grants are exchanged for access tokens. Grant is an abstract concept representing the end-user approval. It can be a code received after the user clicks ‘Approve’ on an access request, or the user’s actual username and password. The original idea behind grants was to enable multiple flows. 1.0 provides a single flow which aims to accommodate multiple client types. 2.0 adds significant amount of specialization for different client type.

**Conclusion**
OAuth 2.0 presents an exalted concept of social sign-on by providing more security than traditional concepts. But, it can be more secure if all the confrontation faced by this protocol are met. Issues like CSRF attack should be construed for developing an enhanced protocol.

**References**


