oAuth Based Single Sign-On

Enabling Seamless Access to MyGov, Associated Sites And Apps

Introduction of oAuth 2.0 based Single Sign-On authentication mechanism in MyGov has enabled better user experience while accessing its various sites and Apps. Now the citizens can engage in seamless participatory activities without multiple time signing in.

ALKA MISHRA
Sr. Technical Director
amishra@nic.in

D.P. MISRA
Scientist-D
dpmisra@nic.in

NARENDER KUMAR JAIN
Scientist-B
nk.jain@nic.in

RAVI KUMAR
Scientist-B
ravi.k@nic.in

Edited by
MOHAN DAS VISWAM

MyGov is one of the most innovative public consultation platforms for deriving various policies and planning in government. This online platform has evolved tremendously since its succesful inception two years ago featuring various kinds of citizen engagement components for better governance. A key highlight of the core technology used in MyGov remains light yet robust whereas various sub-domains viz. Blogs, Newsletters, Volunteering, Survey, SwachhBharat, Innovation, Transforming India and SmartNet besides its mobile Apps.

It is important to provide the user a simple and seamless experience while traversing between MyGov and its various sites. To avoid multiple times signing in by users to access various sites and Apps of MyGov and thus achieve seamless citizen participation/ engagement activities in the platform, enablement of oAuth 2.0 based Single Sign-On (SSO) authentication mechanism has been introduced, which is a standard protocol used by modern social media platforms. With this technology enhancement, a citizen can seamlessly engage in the activities by signing in only once.

oAUTH ROLES:

1. Resource Owner: User

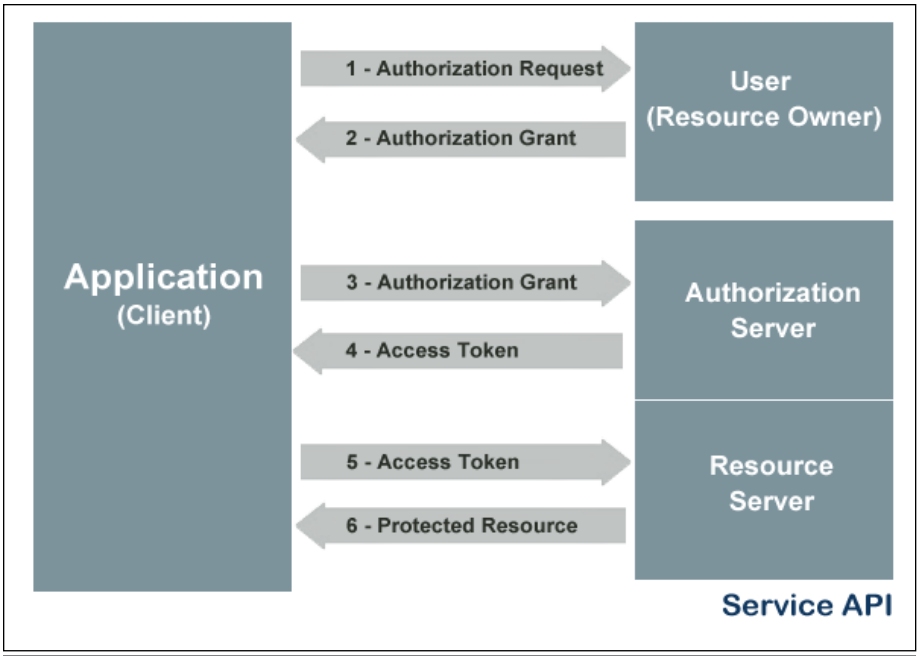
The resource owner is the user who authorizes an application to access their account. The application's access to the user's account is limited to the "scope" of the authorization granted (e.g. read or write access).

2. Client: Application

The client is the application that wants to access the user's account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.

3. Resource and Authorization Server: API

The resource server hosts the protected user accounts, and the authorization server verifies the identity of the user then issues access tokens to the application. From an application developer's point of view, a service's API fulfils both the resource and authorization server roles. Here it is referred to both of these roles combined, as the Service or API.



Abstract Protocol Flow diagram

4. Mechanisms to use MyGov oAuth 2.0 by any client application:

Above diagram depicts the abstract protocol flow of how they generally interact with each other:

APPLICATION REGISTRATION

The application wants to use MyGov oAuth 2.0 will have to be registered with https://auth.mygov.in to get the Client ID and Client Secret Key.

CLIENT ID AND CLIENT SECRET KEY

Once your application is registered, the service will issue "client credentials" in the form of a client identifier and a client secret. The Client ID is a publicly exposed string that is used by the service API to identify the application, and is also used to build authorization URLs that are presented to users. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user's account, and must be kept private between the application and the API.

AUTHORIZATION GRANT

In the Abstract Protocol Flow above, the first four steps cover obtaining an authorization grant and access token. The authorization grant type depends on the method used by the application to request authorization, and the grant types supported by the API. oAuth 2.0 defines four grant types (Authorization Code, Implicit, Resource Owner Password Credentials, Client Credentials) each of which is useful in different cases:

GRANT TYPE: AUTHORIZATION CODE

The authorization code grant type used with server-side Applications is the most commonly used because it is optimized for server-side applications, where source code is not publicly exposed, and Client Secret confidentiality can be maintained. This is a redirection-based flow, which means that the application must be capable of interacting with the user-agent (i.e. the user's web browser) and receiving API authorization codes that are routed through the user-agent.

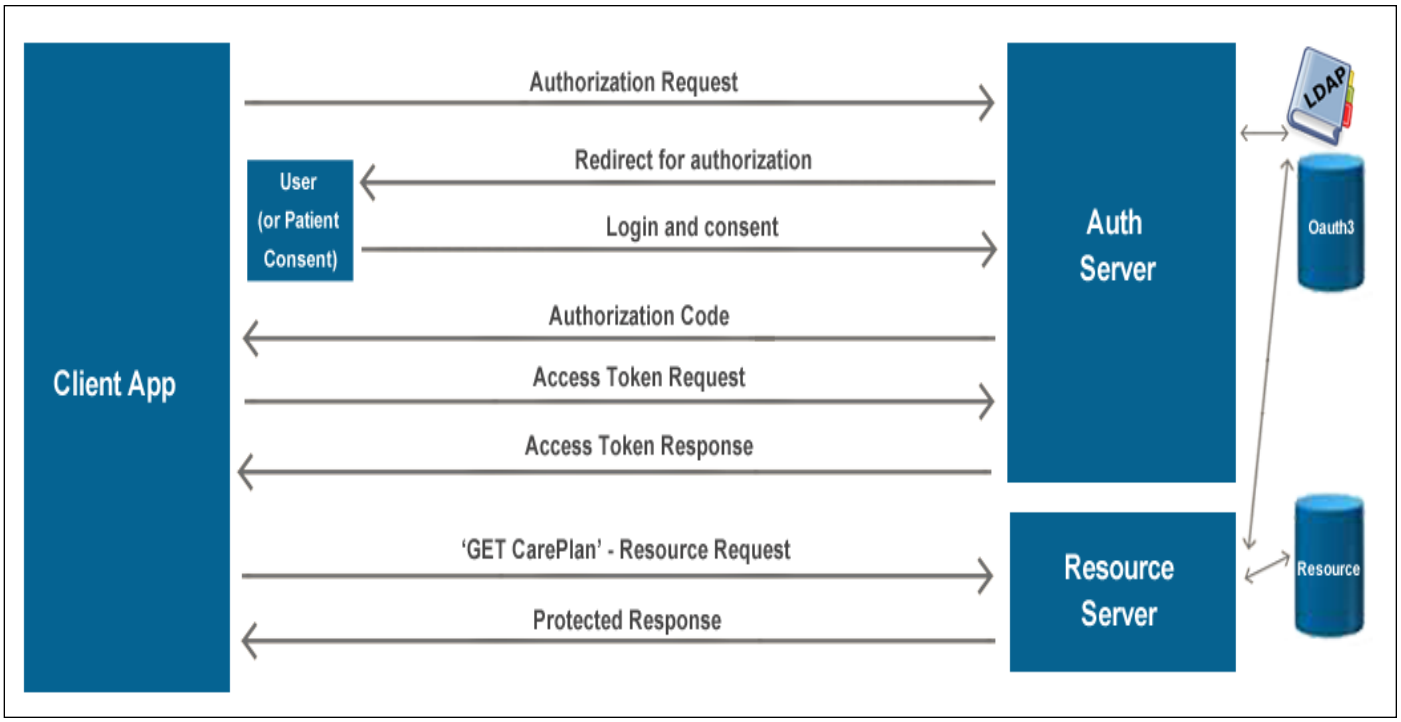
Once the application is authorized! It may use the token to access the user's account via the service API. The service call should be a POST request with Authorisation Headers having the access_token.

GRANT TYPE: IMPLICIT

The implicit grant type is basically used with Mobile Apps or Web Applications (applications that run on the user's device). The implicit grant type is also a redirection-based flow but the access token is given to the user-agent to forward to the application, so it may be exposed to the user and other applications on the user's device.

GRANT TYPE: RESOURCE OWNER PASSWORD CREDENTIALS

With the resource owner password credentials grant type is used with trusted Applications such as those owned by the service itself, the user provides their username and password to the application, which uses the credentials to obtain an access token from the service. This grant type should only be enabled on the authorization server. It should only be used if the application is trusted by the user.



Authorization Code Grant Flow diagram

GRANT TYPE: CLIENT CREDENTIALS

With the Client Credentials grant type used with Applications API access i.e. the user provides their client id and client secret to the application, which uses the credentials to obtain an access token from the service. The application requests an access token by sending its credentials, its client ID and client secret, to the authorization server.

oAUTH SECURITY MODELS

1. Client Impersonation

A malicious client can impersonate another client and obtain access to protected resources if the impersonated client fails to, or is unable to, keep its client credentials confidential.

2. Phishing Attacks

Wide deployment of this and similar protocols may cause end-users to become inured to the practice of being redirected to websites where they are asked to enter their passwords. If end-users are not careful to verify the authenticity of these websites before entering their credentials, it will be possible for attackers to exploit this practice to steal resource owners' passwords

3. Cross-Site Request Forgery

Cross-site request forgery (CSRF) is an exploit in which an attacker causes the user-agent of a victim end-user to follow a malicious URI to a trusting server.

4. Click jacking

In a click jacking attack, an attacker registers a legitimate client and then constructs a malicious site in which it loads the authorization server's authorization endpoint web page in a transparent iframe overlaid on top of a set of dummy buttons, which are carefully constructed to be placed directly under important buttons on the authorization page. When an end-user clicks a misleading visible button, the end-user is actually clicking an invisible button on the authorization page (such as an "Authorize" button). This allows an attacker to trick a resource owner into granting its client access without the end-user's knowledge.

For further information, please contact:
ALKA MISHRA
Sr. Technical Director
MyGov, Platform for Citizen Engagement
379, 3rd Floor, NIC, CGO Complex
Lodhi Road, New Delhi 110 003
Email: amishra@nic.in
Phone: 011-24368854