Creating a google sign-in using OAuth 2.0

Alessandro Tomasi*[1]

[1]Security and Trust, Fondazione Bruno Kessler

December 5, 2018

Abstract

Practice with OAuth 2.0: register a web application with a service provider, log users into your web app, and obtain their consent to access their data held by a federated service provider.

1 OAuth 2.0: authorization code flow

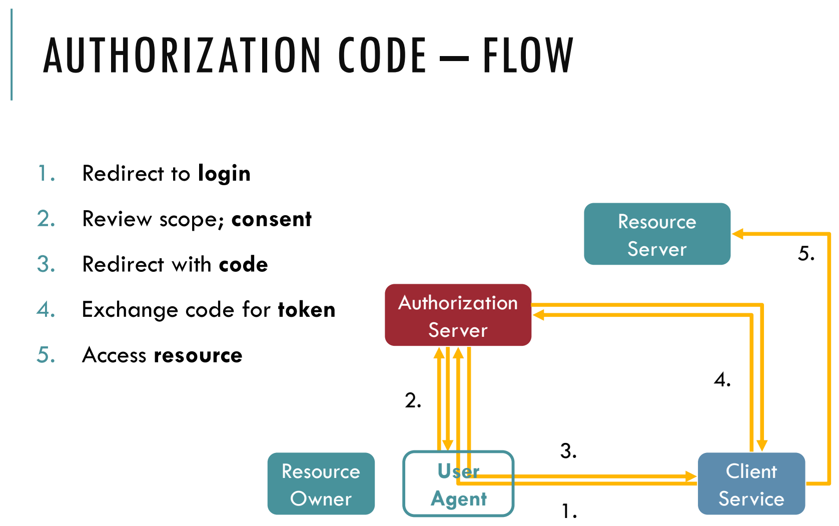
OAuth 2.0 (RFC 6749) is designed on the following premise:

- a resource is being hosted on a Resource Server (`RS`)
- the Resource Owner (`RO`) wishes to grant another client application service (`C`) access to that resource by interacting with an Authorization Server (`AS`) through a User Agent (`UA`), likely a browser.
- The `RO` needs to be registered with the `AS` and authenticate to it with login credentials.
- The `C` needs to be registered with the `AS` and authenticate to it with a client secret.

If `C` may be considered a *confidential* client (e.g. a web app), in the sense that none of the cryptographic material and secret it holds are ever disclosed to other entities, the recommend OAuth flow is the *authorization code flow.* This consists of the following steps:

1. Starting from the `C`, the `RO` is directed through their `UA` to the `AS`'s login page, where `RO` can view the permissions that the `C` is requesting.
2. The `RO` logs in and consents to allowing the `C` access to the resource she owns.

*altomasi@fbk.eu


AUTHORIZATION CODE – FLOW

1. Redirect to **login**
2. Review scope; **consent**
3. Redirect with **code**
4. Exchange code for **token**
5. Access **resource**

Figure 1: OAuth authorization code flow scheme overview.

3. The `RO` is redirected to the `C` with an authorization `code`.
4. The `C` authenticates itself to the `AS` through its previously registered client secret and exchanges the authorization `code` for an access `token`.
5. The `C` can now access the resource by presenting the access `token` to the `RS`.

An overview of the flow can be seen in Figure 1, and a message sequence chart in Figure 2.

2 Set-up

2.1 Obtain OAuth credentials for your app

Google has published generic guidelines for setting up OAuth 2.0. The following notes are specific to setting things up to work with flask on your localhost.

1. Go to the app's page in the Google APIs Console
2. Choose *Credentials* from the menu on the left.
3. Create an OAuth Client ID

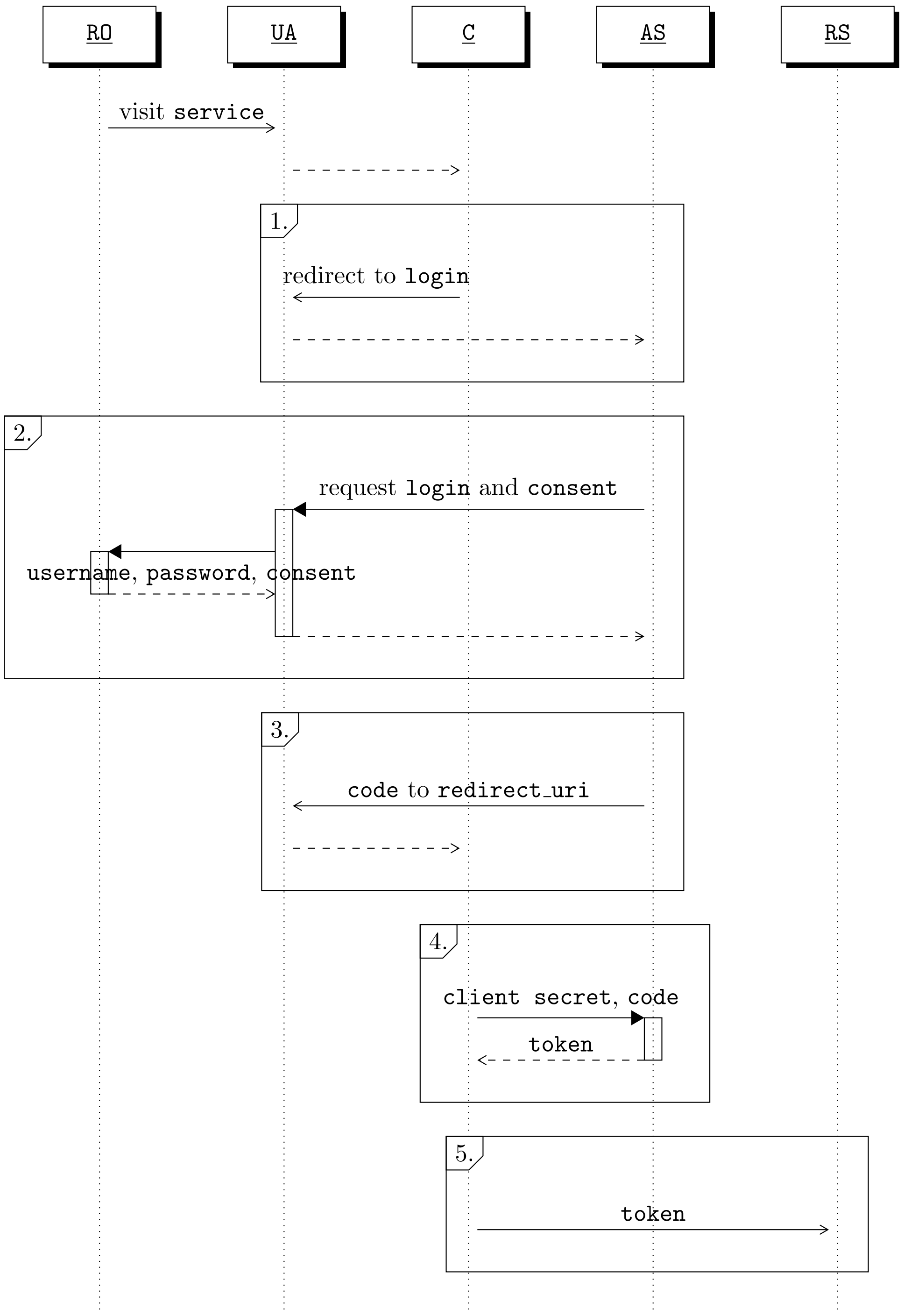


Figure 2: OAuth 2.0 authorization code flow message sequence chart.

4. Configure the consent screen
5. Choose *Web application* as app type
6. Edit client settings: add `http://localhost:5000` to the authorized JavaScript origins and redirect uri
7. You will then be able to get the client ID and client secret.

You can also download the client secret as a JSON data file once you have created it; this will be useful if you choose to manage the authentication flow with google's own libraries.

3 Links

A word of caution: there are many libraries designed to handle oauth flows, with different levels of abstraction. You will have to make your own choice, based not only on preference; for instance, if you want to support many different resource servers it could make sense to use a framework that handles them all; or if you want to use a specific framework like flask, it could make sense to look for something designed specifically for it.

Google provide an overview of the OAuth flows they support, and even provide a number of client libraries, including one for python. They also host an OAuth playground for API calls: explore the available APIs, see concrete examples of exchanged codes and the information returned.

Many outher authorization and resource servers provide third-party apps with guidelines on how to register their clients; for example, dropbox.

Auth0 is an authorization service provider integrating many different services behind the scenes. They also provide a good overview of the Auth0 code flow. They also maintain an auth0-python library.

There is also a generic python library called authlib – see documentation and source.

Lastly, you are not expected to know the details of OAuth flows other than the authorization code flow, but several are described well in this blog post.

4 Exercises

Write your own flask app to handle OAuth authorization code flow to google APIs. There is a minimal working example on github, based on flask-oauth.

1. Register your flask web app with google, obtaining OAuth credentials – client id and secret.
2. Set a redirect uri, probably something beginning with `http://localhost:5000`, or whatever is appropriate for your app. Assuming this works for you, ask yourself: why does it work, and should it work?

3. Create OAuth endpoints in your app: at a minimum I would suggest a route for the user to visit to start the process, and one to serve as redirect uri.

4. Go through the login process with a google account. Specify the scope you prefer for the request, such as profile or email.

5. Go to your google account page and verify that your app is listed under those having access, now you have given consent.

6. Once you have successfully managed to log in, try to log out.

7. Try to repeat the above steps with a different authorization server, e.g. dropbox or facebook.

Optionally, use `wireshark` to check that your traffic really is in `https`, and neither the user's login details or the client secret are being exposed. Remind yourself of the `SSL` handshake protocol in the process.