Universita degli Studi di Parma

DIPARTIMENTO DI INGEGNERIA ED ARCHITETTURA

Corso di Laurea di ingegneria Informatica, Elettronica e delle Telecomunicazioni

---

OAuth2

Relatore:
Prof. Ing. Poggi Agostino

Tesi di laurea di:
Corradi Alessandro

A.A. 2019/2020

Contents

Introduction

The problem

Let's imagine that we have a server that manages a vast list of products. Those products could be searched, filtered and sorted by different fields, like the name, color, insertion date and so on. The search query is submitted, the server returns a web page with the products, represented in a tabular fashion.

The server becomes popular and is used by million of users, and you start to think that sharing those search results could be a cool feature.

We want to use third party API to integrate this feture.

Different ways to achieve the goal, here are a few of them:

Direct transmission of the user identifier

Directly send the user id to the server

```
$ curl -v -X POST http://localhost:8080/db_auth '{"user-id": 1}'
```

I do not want to sped too much time on this one. It is for sure one of the fastest and less secure way to communicate the user identity.

Even someone with some basic knowledge of programming could recognize that by changing the value of `user-id`, the request would still be valid and you are recognized as another user.

Http Basic Authentication

Http Basic Auth: each API request is signed with a username and a passwrod, encoded in base64

```
$ curl -u Aladdin:OpenSesamus -v http://localhost:8080/basic

> GET / HTTP/1.1
> Host: localhost:8080
> Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW11cw==
> User-Agent: curl/7.72.0
> Accept: */*
...
```

Since the credentials are neither hashed nor encrypted, your username and password are always visible by someone who could read the headers. Therefore this protocol requires an https channel in order to be somewhat secure.

Public/Private Key

```
$ ssh-keygen -v
The key fingerprint is:
SHA256:tFKUq1HLfJ3Iy6toGec804dKXPgx4P6rrL6Tc4S0wVY ale-cci@vagrant

The key's randomart image is:
+---[RSA 3072]----+
|        ..       |
|       .E        |
|     . =++ o .   |
|      *+=++ o    |
|     o.*So+.     |
|      =+oooo     |
|       O+..o     |
|      *+*.+ .    |
```

```
|     o=*=*oo     |
+----[SHA256]-----+
```

```
$ ssh-copy-id -i ~/.ssh/id_rsa username@host
```

Authorization via asymmetric cryptography. Two keys are generate a public and a secret one. The public key is, as the name suggests, public and everyone could read it. The client encodes his messages with the private key; the receivers could validate the authenticity of those messages using the public key.

This is one of the safest methods, but it's not viable as we need to distribute in some way the private key to trusted users.

OAuth2

Provide an access token at each client, which they could use to validate their identity at each API request. This token could be either valid, expired, revoked or invalid.

```
$ curl -v --header 'Authorization: Bearer 1234' http://localhost:8080/api

> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.72.0
> Accept: */*
> Authorization: Bearer 1234
...
```

The OAuth2 protocol provides a secure and standardized way for creating and exchanging those token between client and server.

OAuth2 Protocol

Obtain an access token

1. Redirect to the authorization provider.

   ```
   http://google.apis.com?response_type=code&client_id...
   ```

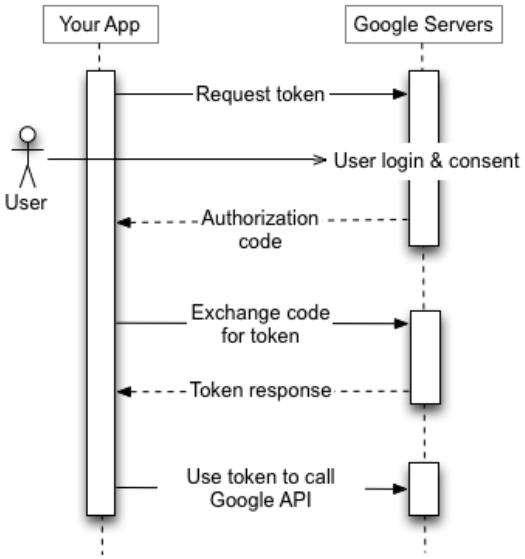
2. If the authorization is successful, the oauth server will redirect to `redirect_uri` passing a `code` as GET parameter. Otherwise `error` contains the reason why the authorization was not successful.

   ```
   POST /oauth/token HTTP/1.1
   Host: authorization-server.com

   grant_type=authorization_code
   &code=xxxxxxxxxx
   &redirect_uri=https://example-app.com/redirect
   &client_id=xxxxxxxxxx
   &client_secret=xxxxxxxxxx
   ```

   If an authorization code is used more than once, the authorization server must deny the subsequent request.

3. The client exchanges the `code`, called also "grant token", with the server to obtain an `access_token`.
4. The server returns the access token with additional informations, such as expire date and JWT.
5. The client uses the access token for APIs requests.



Revoke an access token

Something something

SSO with OpenID

Second problem

We want to delegate the process of authentication of username and password to an external service, then if the user is registered in our platform we allow him in.

JWT

Shorthand JSON Web Token, encoded token that contains the necessary informations to identify the user.

JWS e JWE

Token could be signed, encrypted or both.

JWS Validation

Verify the SHA256 signature

References

[1] Using OAuth 2.0 to Access Gooogle APIs
`https://developers.google.com/identity/protocols/oauth2`