Introduction to OAuth2

About the Speaker

- Java developer, recently doing more in node.js :)
- Working at hybris (now an SAP company)
- Involved in OAuth 2.0 & OpenId Connect 1.0 in YAAS (https://www.yaas.io)

Agenda

- What and Why.
- How does it work?
- Further steps
- Q&A

What and Why

OAuth 2.0

- Is a specification (http://oauth.net/2/)
- Replaces previous specification: OAuth 1.0
- based on HTTP
- Is not a protocol/framework

The Definition (from the spec)

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 [...], or by allowing the third-party application to obtain access on its own behalf.

What and Why

Q: OK… what’s it all about then?

A: It’s all about **delegating authorization**

(in a ***safe*** way)

Few Examples

- “Real life”:
  - access cards
  - valet parking keys
- IT world:
  - OS-level permissions (Android, iOS)
  - Google Drive integration: e.g. **Drive Notepad**.

What and Why: Summary

OAuth 2.0 enables third-party applications to access user data securely, without requiring the user to take the “scary step” of passing over the account password.

User remains in control of which resources (and operations) are available to every authorized application.

How does it work: Basic Principles

- Use different set of credentials (than those of the user) called **Access Token**
- Access Tokens specify scope, lifetime and other access attributes
- Application uses an Access Token (instead of user credentials) to access resources

Four Roles

- *resource owner* - end user
- *resource server* - the server hosting protected resources, “understands” Access Tokens
- *authorization server* - the server issuing Access Tokens to clients
- *client* - third-party application making requests to protected resources. **Had to be registered within authorization server**

Access Tokens

- Credentials used to acces protected resources
- Primary construct to replace username/password - based authorization
- Short-lived, e.g. one hour.
- We only see just opaque String, e.g. “da5f548fce7e351149cb…”
- Passed in HTTP Authorization header, but query param and body possible.

An example header value: “**Authorization: Bearer da5f548fce7e351149cb…**”

Access Tokens, continued

- Carry information about:
    - resource owner
    - client (app)
    - scopes
    - expiration
    - others? (implementation dependent)

Scopes

- Define **authorization scope**.
- Can be used to model different kinds of “permissions”
- Not pre-defined by specification

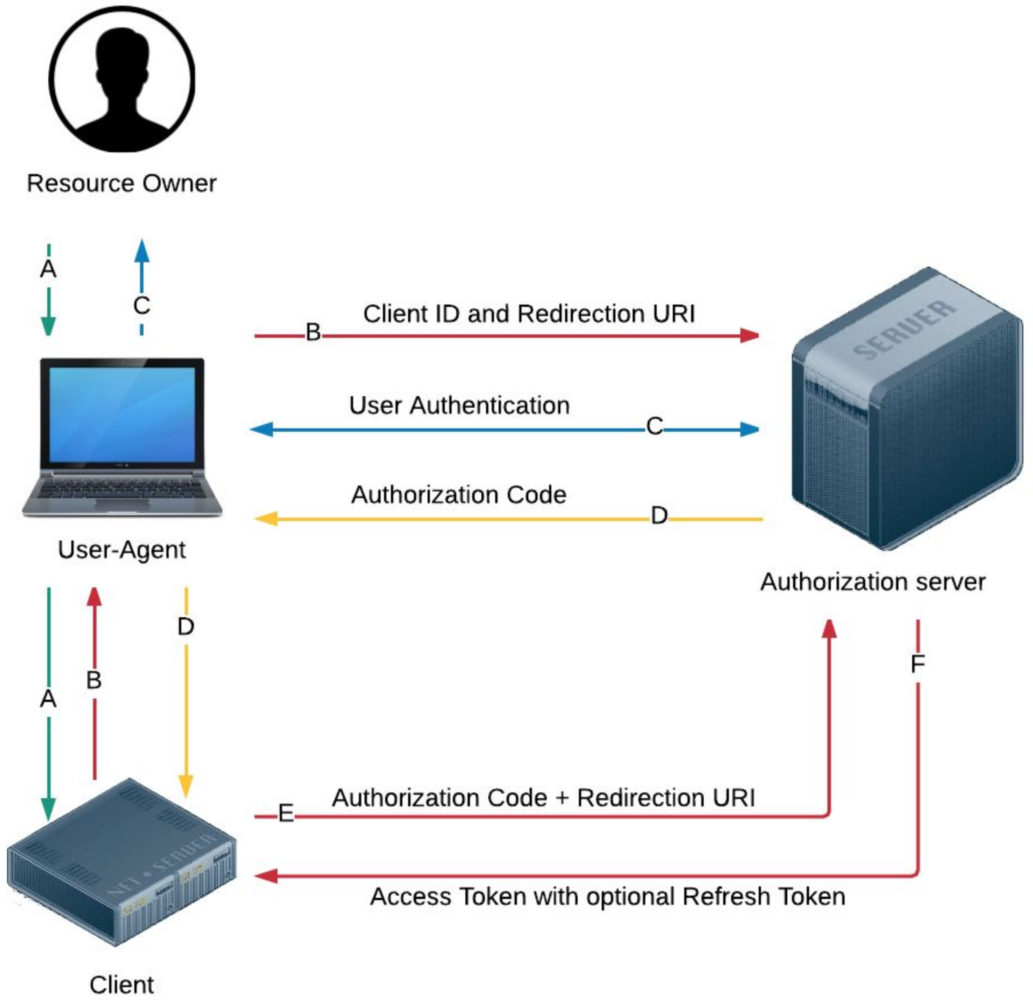
Four Grant Types

- Authorization Code
- Implicit
- Client Credentials
- Resource Owner Password Credentials

Authorization Code Grant

- Also called “Server-Side Web Applications Flow”
- The most secure one
- Target client: web server applications (with secure server-side)
- Intended to keep both **user credentials** and **access tokens** secure
- Can use refresh tokens (if supported)

Authorization Code Grant


Resource Owner
A
C
Client ID and Redirection URI
B
User Authentication
C
Authorization Code
D
User-Agent
Authorization server
A
B
D
F
Authorization Code + Redirection URI
E
Access Token with optional Refresh Token
Client

Authorization Code Grant: an Exercise

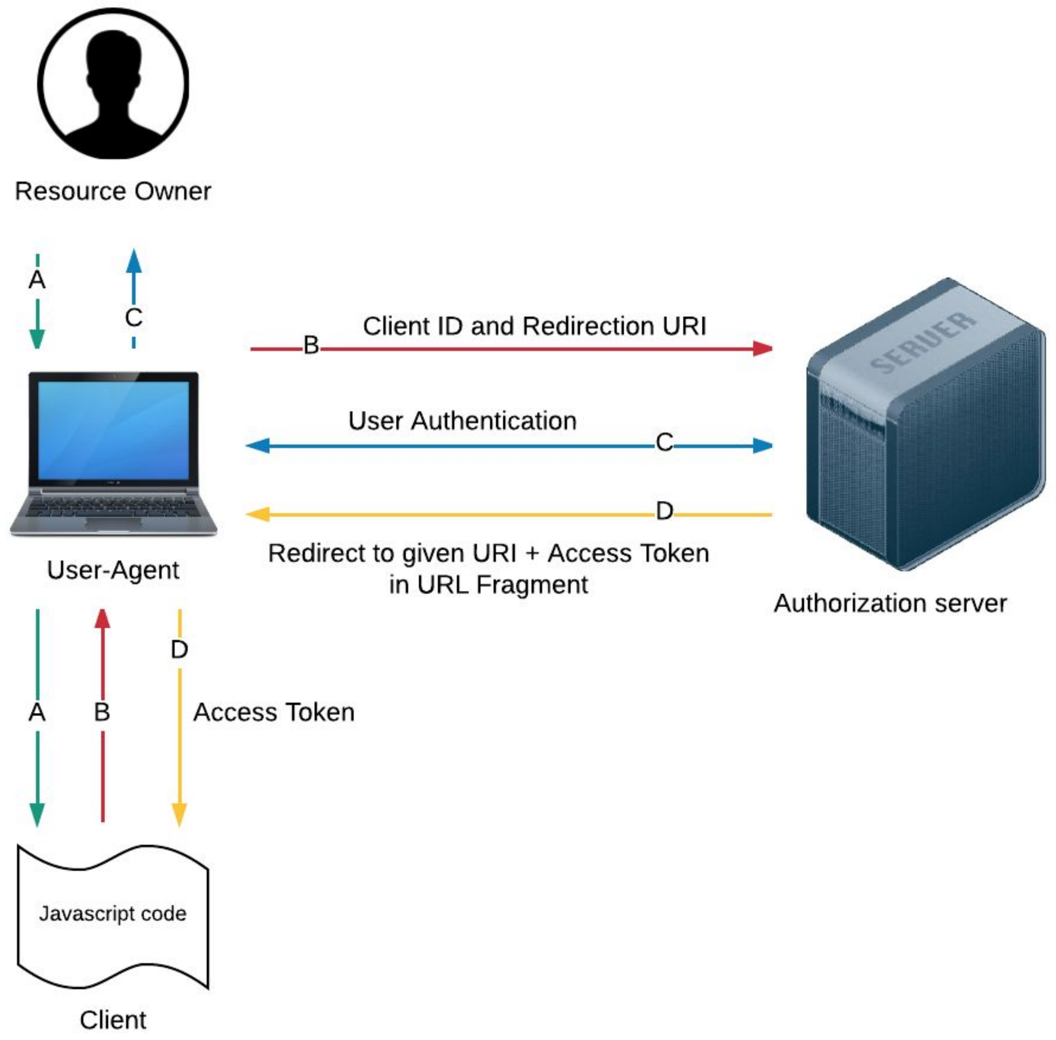
Using Authorization Code Grant in GitHub

- Client Registration Page in GitHub:
- Authorization Code Grant demo: Creating new repository

Implicit Grant

- Also called “Client-side Web Applications Flow”.
- Designed for user-agent based applications (no server side)
- Less secure (but considered “secure enough”)
- Intended to keep **user credentials** secure
- Access Tokens are visible to user-agent (browser) code
- No client identification (so called *public clients*)
- No refresh tokens

Implicit Grant



Implicit Grant: an Exercise

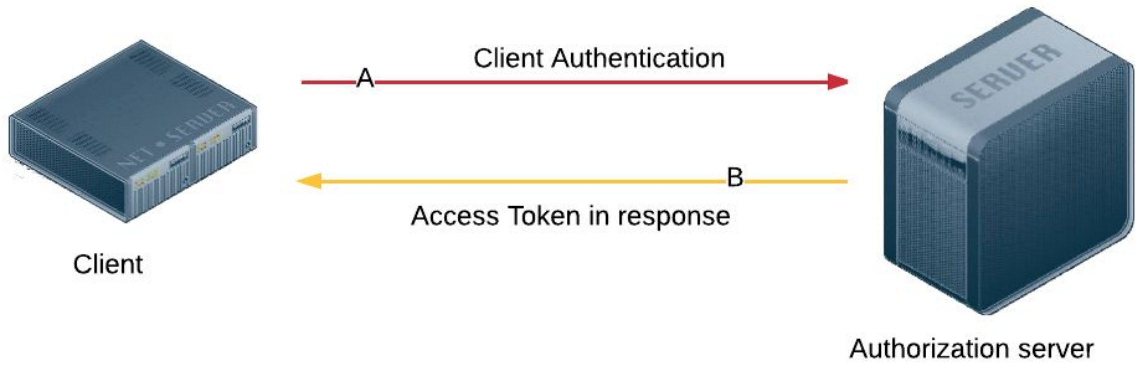
Using Implicit Grant in Google

- Client Registration Page in Google
- Implicit Grant demo: Getting public profile

How does it work: Client Credentials Grant

- No resource owner involved
- Useful for batch jobs or service to service calls
- May require pre-authorization (no consent page!)

Client Credentials Grant



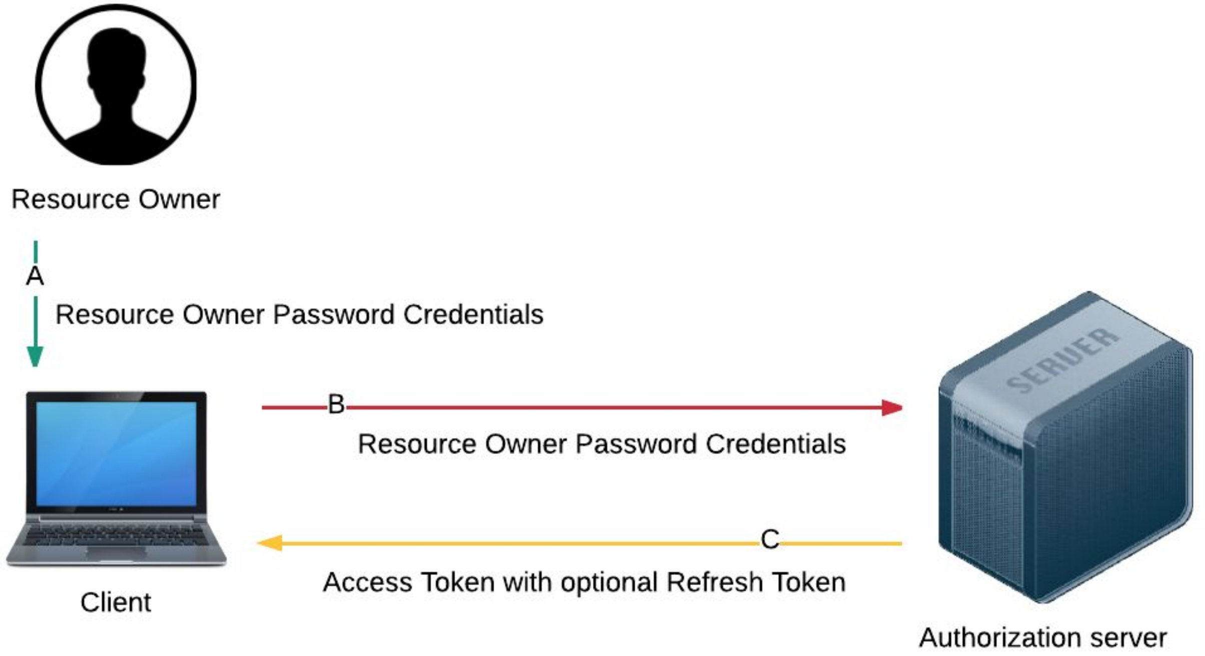
Example:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"example",
    "expires_in":3600,
    "example_parameter":"example_value"
}
```

Resource Owner Password Credentials

- Based on username/password, like on old client-server tools.
- Still better than old tools, because client can immediately “forget” the password and use Access Token instead
- For scenarios where user controls the client: CLI tools, desktop apps, etc.
- Can use Refresh Tokens (to avoid user having to re-enter the password)

Resource Owner Password Credentials


Resource Owner
A
Resource Owner Password Credentials
B
Resource Owner Password Credentials
C
Access Token with optional Refresh Token
Client
SERVER
Authorization server

Example:

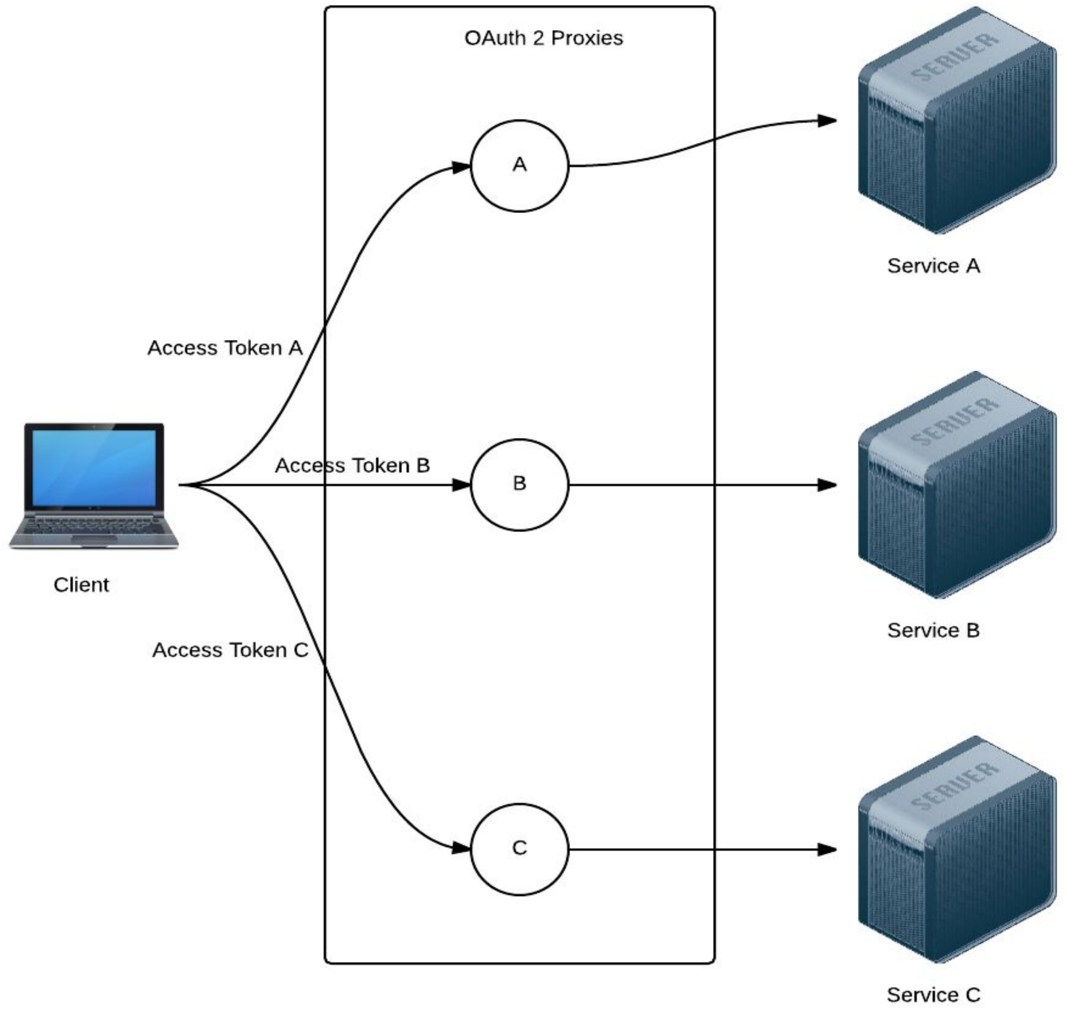
```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic
czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"example",
    "expires_in":3600,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
    "example_parameter":"example_value"
}
```

Client Credentials Grant: an Exercise

Using Client Credentials Grant in YAAS

- Client Credentials Grant demo: Playing with HelloToad

API Proxy concept



Further steps

- Specification: http://tools.ietf.org/html/rfc6749
- Tutorials: http://tutorials.jenkov.com/oauth2/index.html
- “Hello, World” apps: Google
    - docs: https://developers.google.com/identity/protocols/OAuth2?hl=en
    - developer’s console: https://console.developers.google.com
    - Google OAuth playground: https://developers.google.com/oauthplayground
    - Authorized apps: https://security.google.com/settings/security/permissions

Get started in YAAS

https://devportal.yaas.io/gettingstarted/

- Cloud platform for building commerce solutions
- You can register for free (hosting plan included)
- OAuth 2.0 & OpenID Connect 1.0 providers to secure your API’s
- We’re in Beta now, but we go public soon

Explore the OAuth 2.0 ecosystem

https://www.oauth.io

- Claim to provide access to over 100 OAuth 2.0 providers
- Easy to use SDK API to hide OAuth 2.0 provider - specific implementation details
- Commercial service, free plan will allow only for 2000 requests :(

References

- Book:“Getting Started with OAuth 2.0”, by Ryan Boyd, O'Reilly Media, Inc., ISBN: 9781449311605

- OAuth 2.0 home page: http://oauth.net/2/
- OAuth 2.0 RFC: http://tools.ietf.org/html/rfc6749
- OAuth 2.0 RFC Errata: http://http://www.rfc-editor.org/errata_search.php?rfc=6749
- OAuth 2.0 Authorization Framework: Bearer Token Usage: http://tools.ietf.org/html/rfc6750
- OAuth 2.0 Threat Model and Security Considerations: https://tools.ietf.org/html/rfc6819

- Open ID Connect 1.0 website: http://openid.net/connect/
- Open ID Connect Core RFC: http://openid.net/specs/openid-connect-core-1_0.html

- **YAAS: www.yaas.io**

- Google developers console: https://console.developers.google.com
- Google OAuth playground: https://developers.google.com/oauthplayground

Q & A