Sécurité sociale

OAuth2 Integration Client Credential

Dimona REST Web Service

DOCUMENT HISTORY

Version	Date	Author	Description of changes / remarks
0.01	04/12/2019	Florent Marteau	Draft version.
0.1	04/12/2019	Stéphane Flamme	Review
0.2	30/06/2022	Florent Marteau	New OAuth v4 URLs
0.3	31/03/2023	Aymerick Soyez	New OAuth v5 URLs
0.4	04/03/2024	Aymerick Soyez	Add “Dimona only” note

TABLE DES MATIERES

1. Présentation du protocole OAuth 2.0

OAuth 2.0 est un protocole de sécurité permettant à une personne de déléguer ses droits d'accès à une ressource. Ce protocole permet à une application d'agir au nom de quelqu'un sans devoir fournir les informations secrètes de cette personne. Ce protocole est utilisé pour sécuriser les web service REST.

L'application cliente récupère auprès d'un serveur d'authentification un Access Token qui va lui permettre d'accéder à une ressource protégée au nom du propriétaire de la ressource.

1.1. Lexique

Ci-dessous se trouvent les termes spécifiques à OAuth.

- ***Resource*** : est quelque chose dont on veut protéger l'accès et pour lequel le Resource owner peut choisir de déléguer les droits d'accès.
  OAuth ne définit pas la nature de la ressource. Il est donc possible de construire un système de gestion des droits d'accès à des applications. Dans ce cas, la ressource est le droit d'accès.
- ***Token :*** est un objet obtenu auprès de l'Authorization Server qui permet de prouver que le propriétaire de la ressource nous a confié l'accès à celle-ci. OAuth définit deux types de Token différents :
  - **Access Token** utilisé pour accéder à une ressource
  - **Refresh Token** permettant de renouveler un Access Token dans certains flux d'autorisation sans nécessiter la présence du Resource owner
- ***Scope :*** Permet de définir les droits d'accès à la ressource délégués par le Resource owner. Par exemple, avec le scope "read", une personne peut accéder à la ressource mais ne peut pas la modifier.

1.2. Acteurs

- ***Resource owner**:* Le propriétaire de la ressource qui veut autoriser une application à agir en son nom. Généralement, celui-ci est une personne.
- ***Protected resource** :* La ressource protégée auquel le propriétaire a accès. Celle-ci peut avoir différentes formes mais généralement c'est une web API qui peut avoir différents droits d'accès (lecture, écriture,…)
- ***Client** :* Le client est l'application qui accède à la ressource protégée au nom du propriétaire de cette ressource. Cette application peut par exemple être une application située sur un serveur, une application javascript ou native.
- ***Authorization server** :* Le serveur d'autorisation permet de fournir un A*ccess Token* que le client peut utiliser pour agir à la place du propriétaire de la ressource. Il va pour cela authentifier le propriétaire et vérifier son accord.

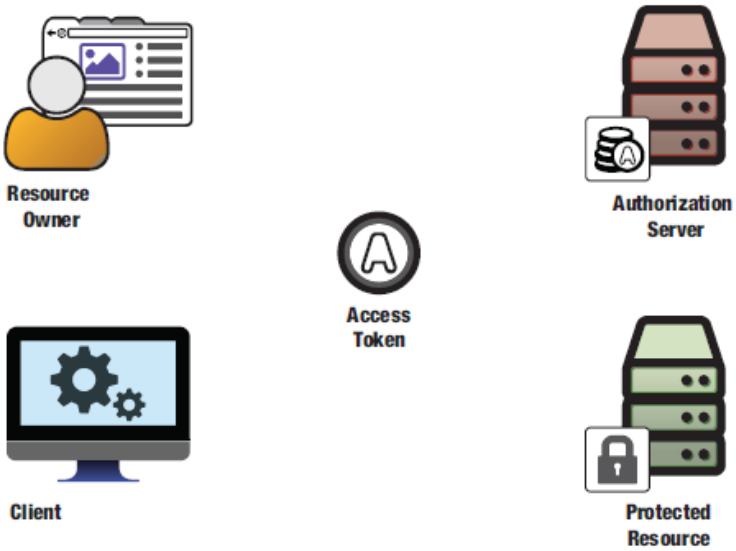


Figure 1 Les acteurs OAuth 2.0

1.3. Etapes du processus d'autorisation

Dans le cadre de l'utilisation du flux client credential, une application cliente n'agit pas au nom d'une tierce partie mais en son nom propre. Dans ce scénario, le client est donc le Resource Owner.

1.3.1. Identification et authentification du client

Au départ du processus, le Resource owner contacte le Client (1) et lui signale qu'il aimerait que celui-ci agisse en son nom. Suite à cette demande, le Client contacte l'Authorization server (2) et indique qu'il veut pouvoir agir à la place du Ressource owner. Pour cela, le client met le Resource owner en contact avec l'Authorization server en transmettant différentes information (3) permettant d'identifier les droits demandés sur la ressource (via le scope) ainsi qu'une URL permettant à l'Authorization server de contacter le client une fois la demande accordée.

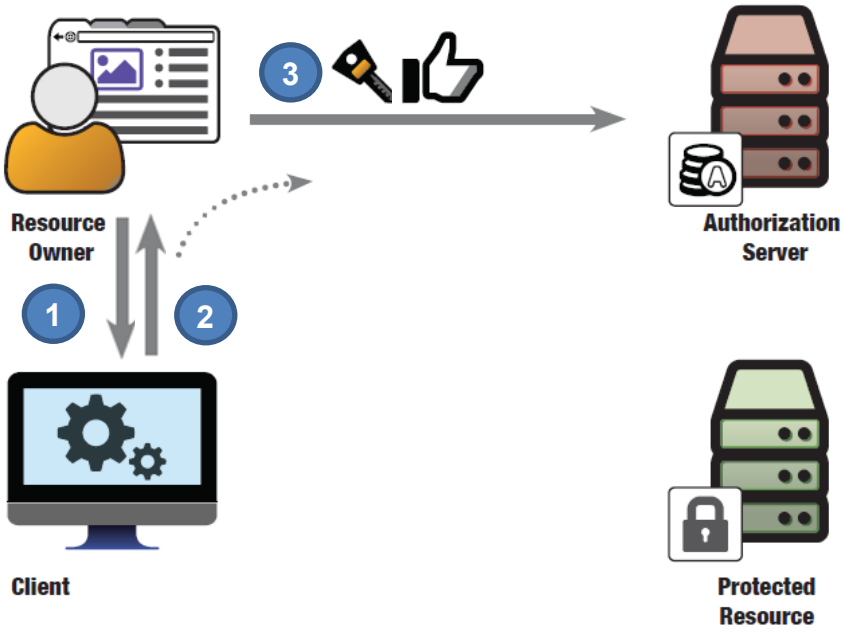


Figure 2 Le Client redirige le Resource owner vers l'Authorization server pour authentification et accord

1.3.2. Demande de token auprès du serveur d'autorisation

Une fois le Ressource owner identifié et authentifié auprès de l'Authorization server, celui-ci demande l'approbation du Resource owner (1) en précisant les droits (scope) demandés par le client sur la ressource. Une fois l'approbation obtenue, l'Authorization server redirige le Resource owner vers le client (2) en utilisant l'URL de redirection obtenue précédemment.

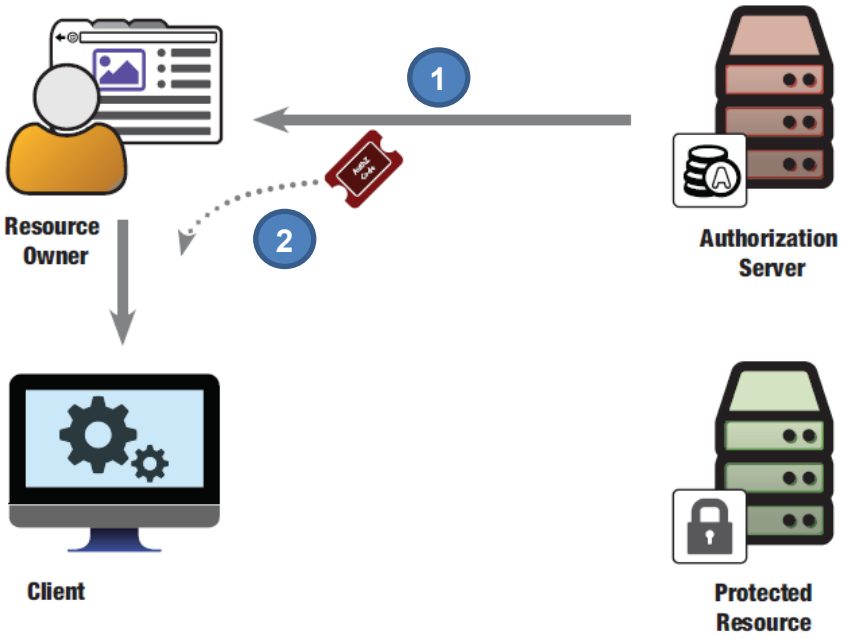


Figure 3 L'Authorization server transmet un Authorization Code au Client

1.3.3. Accès à une ressource protégée

Une fois le Client en possession d'un Access Token et ceci tant que l'Access Token reste valide, il peut utiliser celui-ci pour accéder à la ressource. Pour ce faire, il va transmettre son identifiant, son secret et l'Access Token au Resource server (1). Celui-ci contacte l'Authorization server en transmettant ces informations (2) afin que celui-ci vérifie la légitimité de la demande. Une fois la demande approuvée (3) par l'Authorization server, le Resource server retourne au Client la ressource demandée.

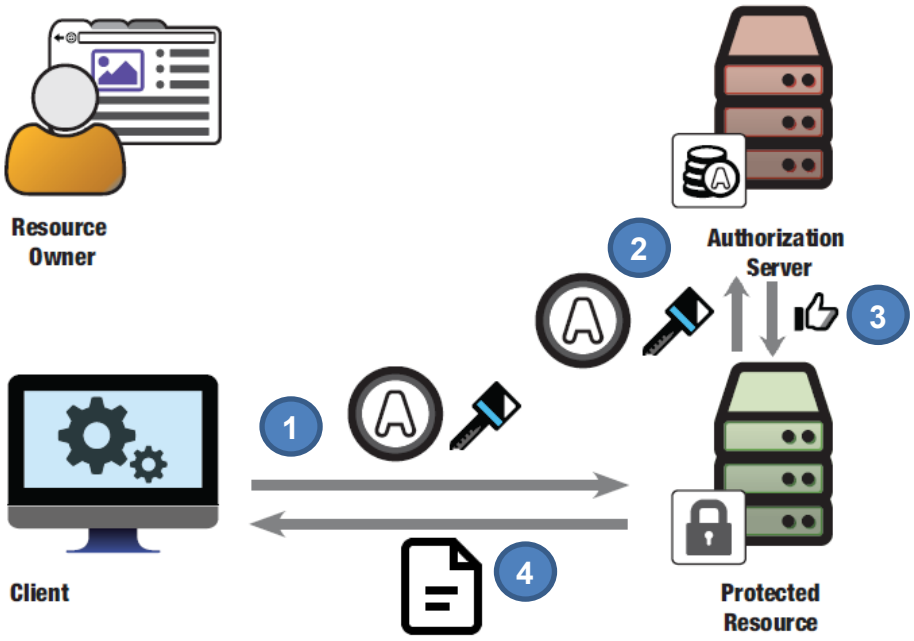
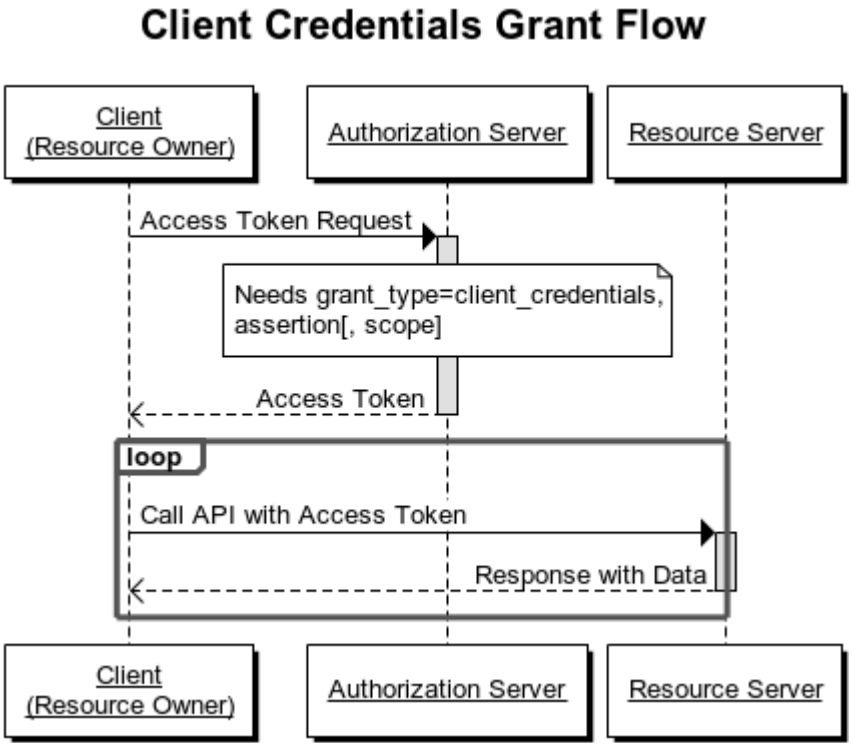


Figure 4 Utilisation d'un Access Token pour obtenir une ressource protégée par OAuth

2. Intégration client credential

Cette section présente plus spécifiquement le flux client credential pour le Web Service REST Dimona. Ce flux est utilisé lorsque le client agit pour son propre compte. Ce flux permet l'authentification de personne morale plutôt que de personne physique. Ce flux est donc utilisé pour les entreprises ou les prestataires qui voudraient accéder à une ressource REST en leur nom.
Dans cette partie, vous trouverez les informations nécessaires afin de créer votre requête pour appeler le serveur d'autorisation ainsi que le format de réponse possible.



2.1. Client Credentials grant flow

Le client peut demander un access token en utilisant uniquement ses credentials (ou un autre moyen d'authentification supporté) quand le client demande un accès à une ressource protégée sous son contrôle ou demande l'accès à des ressources protégées sur la base d'une autorisation préalablement accordée par le serveur d'autorisation.

Le flux client credential doit uniquement être utilisé par des clients confidentiels.

2.1.1. Requête d'autorisation et réponse

Etant donné que l'authentification du client est utilisée en tant qu'attribution d'autorisation, aucune demande d'autorisation supplémentaire n'est nécessaire.

2.1.2. Access token request

Le client effectue une demande au token endpoint en ajoutant les paramètres suivants utilisant le format "application/x-www-form-urlencoded" avec un encodage UTF-8 dans le corps de la requête HTTP. Ces paramètres sont directement tirés de la RFC 7519 :

grant_type
REQUIS. La valeur doit être « client_credentials ».

scope
OPTIONNEL. Le scope de la requête.

client_assertion_type
REQUIS. La valeur doit être « « urn:ietf:params:oauth:client-assertion-type:jwt-bearer »

client_assertion
REQUIS. Un JWT signé qui authentifie le client auprès du serveur d'autorisation comme décrit dans RFC 7519. Le contenu du JWT est décrit dans la section « Client authentification ».

Voici un exemple de requête HTTP :

POST /REST/oauth/v5/token HTTP/1.1

```
Content-Type=application/x-www-form-urlencoded
charset=UTF-8
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-
type%3Ajwt-
bearer&grant_type=client_credentials&scope=scope%3Awarlock%3Atest%3Arest%3Aappl
ication&client_assertion=eyJhbGciOiJSUzI1NiJ9.eyJhdWQiOiJodHRwczpcL1wvb2F1dGhpb
nQuc29jaWFsc2VjdXJpdHkuYmUiLCJzdWIiOiJ3YXJsb2NrOnRlc3Q6d2ViOjEiLCJpc3MiOiJ3YXJs
b2NrOnRlc3Q6d2ViOjEiLCJleHAiOjE1Mzk2NzQ2NjEsImlhdCI6MTUzOTY3NDYxMSwianRpIjoiNGI
0ZmIyMjIyNjE1ZGNhMTA4MGMzOWQ2YTVhMjFmYzAxZTI5Yzk5NGVjNjU5OTUwMDdlNDdhYzJkOTRhMT
liZjQzZjhmYzg0ZWZmYmJhOTA0NjM1YjZjMGZkYjE1NmI5YWFmYTQ5ZTFhYzE4MmFlYWY1MGIwMDY3M
zc4ZWJlZTIifQ.hRnJs5I4HVlC_YwQM6zVq0vj1JhuczTdweoLjfUnFmGusFSkDKQXUpqZMbi8LOwxy
eIyIszO3uy6TEOwZbyzp57Ms4YS2OmyQrF_3wvblM5rIkTxLBdsKoOxvIjyZKWmopy1V3CKUhtOL8UL
JeMquX4BHF1PBW4rRYJjGzhMKMZ5RFO-
wwOmzoa6VrMPpCsVpaKb0BPexGcNQJHyT2gjyLd2QZ2Ij5ctrxq7TEOn4bk0RvYyM8xxXQ70WAl7QsC
sZCYjKz55yGmdb9YKBU__i1uIOlELa0DtKrK3vP2SJxnVIBeC9IA_MWya9sBwqCyW8WS7dmr6UHpwGl
EWGr-u_Q
```

2.1.3. Access token response

Si la requête d'access token est valide et autorisée, le serveur d'autorisation fournit un access token. Un refresh token n'est jamais fourni dans le flux client credential. Si la requête échoue ou est invalide, le serveur d'autorisation renvoie une réponse avec une erreur.

La réponse contient les paramètres suivants tires de la RFC 7519 :

access_token
L'access token fournit par le serveur d'autorisation.

token_type
Type du jeton émis comme décrit dans RFC6749 section 7.1.

expires_in
La durée de vie en seconde de l'access token. Par exemple, la valeur "3600" dénote que l'access token expirera une heure après la génération de la réponse.

Exemple de réponse correcte :

```
HTTP/1.1 200 OK
Date: Tue, 16 Oct 2018 09:37:56 GMT
Server: Apache
Content-Length: 298
Content-Type: application/json; charset=UTF-8
{
    "access_token":"7hbq6oh04a8h5asq1kon18fl4m",
    "scope":"scope:warlock:test:rest:application",
    "token_type":"Bearer",
    "expires_in":43199
}
```

2.1.4. Error response

Le serveur d'autorisation répond avec un statut http 400 (mauvaise requête) et inclus les paramètres suivants tirés de la RCF 7519 :

error
REQUIS. Un code d'erreur unique ASCII [USAASCII] parmis les suivants :

invalid_request
Il manque un paramètre obligatoire dans la requête, une valeur non-supportée pour un paramètre (autre que le grant type), un paramètre répété. La requête utilise plus d'un méchanisme pour authentifier le client ou est mal formée.

invalid_client
L'authentification du client a échouée (client inconu, authentification du clien non-incluse ou méthode d'authentification non-supportée).

invalid_grant
Le flux d'authentification utilisé est incorrect.

unauthorized_client
Le client authentifié n'est pas autorisé à utiliser ce type de flux.

unsupported_grant_type
Le flux n'est pas supporté par le serveur d'autorisation.

invalid_scope

Le scope demandé est invalide, inconnu, malformé ou n'est pas offert par le propriétaire de la ressource.

error_description
OPRIONNEL. Message fournissant des informations complémentaires sur l'erreur survenue.

error_uri
OPTIONNEL. Une url fournissant une page web avec une documentation de l'erreur obtenue.

Exemple d'une réponse erronnée:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
        "error":"invalid_request",
        "error_description":"Request was missing the client_id parameter.",
        "error_uri":"https://tools.ietf.org/html/rfc6749",
}
```

2.2. Client Authentification

Un JWT signé qui authentifie le client au sein du serveur d'autorisation, comme décrit dans la RFC 7519.
Le contenu du JWT est décrit dans la section « client authentification » de la RFC :

jti
REQUIS. Ce paramètre fournit un identifiant unique pour le JWT Ce paramètre est sensible à la casse des caractères.

iss
REQUIS. Ce paramètre identifie l'applicatif qui fournit le JWT. Ce paramètre est sensible à la casse des caractères.

sub
REQUIS. Ce paramètre identifie le principal qui est le sujet du JWT. Ce paramètre est sensible à la casse des caractères, contenant une valeur string ou URI.

aud
REQUIS. Ce paramètre définit le destinataire du JWT..

exp
REQUIS. Ce paramètre définit la date d'expiration à laquelle ou après laquelle le JWT ne doit plus être accepté.

nbf
REQUIS. Ce paramètre identifie le timestamp avant lequel le JWT ne peut pas être accepté.

iat
REQUIS. Ce paramètre définit le timestamp auquel le JWT a été créé.

3. URLs

URL du serveur : **https://services.socialsecurity.be/**
Point d'entrée :

- Back channel: **https://services.socialsecurity.be/REST/oauth/v5/token**

Audience : **https://services.socialsecurity.be/REST/oauth/v5/token**