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

INHALT

1. VORSTELLUNG DES PROTOKOLLS OAUTH 2.0 3

1.1. Glossar 3

1.2. Etappen des Autorisierungsprozesses 5

2. INTEGRATION DER CLIENT CREDENTIALS 8

2.1. Client Credentials grant flow 9

2.2. Client Authentification 12

3. URLS 13

1. Vorstellung des Protokolls OAuth 2.0

OAuth 2.0 ist ein Sicherheitsprotokoll, mit dem eine Person ihre Zugriffsrechte einer Ressource übertragen kann. Dieses Protokoll erlaubt einer Anwendung, ohne Angabe der geheimen Daten einer Person, im Namen dieser Person zu agieren. Dieses Protokoll dient als Sicherheitsmaßnahme für die REST-Webservices (web service REST).

Die Client-Anwendung erhält von einem Authentifizierungsserver ein Access Token, mit dem sie im Namen des Besitzers einer geschützten Ressource Zugriff auf diese Ressource erhält.

1.1. Glossar

Im Folgenden werden einige OAuth-spezifische Bezeichnungen erläutert.

- ***Resource*** (Ressource): etwas, zu dem man den Zugriff absichern möchte und für das der Resource owner Zugriffsrechte erteilen kann.
  OAuth definiert die Art der Ressource nicht. Es ist deshalb möglich, ein Verwaltungssystem von Zugriffsrechten für Anwendungen aufzubauen. In diesem Fall ist die Ressource das Zugriffsrecht.
- ***Token:*** ein Objekt, das man vom Authorization Server erhält und das als Nachweis dafür dient, dass einem der Besitzer der Ressource das Zugriffsrecht zu dieser Ressource erteilt hat. OAuth definiert zwei verschiedene Typen von Token:
  - **Access Token** werden für den Zugriff auf eine Ressource verwendet**.**
  - **Refresh Token** dienen zur Erneuerung eines Access Token in manchen Autorisierungsabläufen, ohne dass der Resource owner anwesend sein muss.
- ***Scope:*** Umfang der durch den Resource owner erteilten Zugriffsrechte auf die Ressource. Mit dem Scope „read“ beispielsweise kann eine Person auf die Ressource zugreifen, kann sie aber nicht ändern.

Rollen

- ***Resource owner**:* Besitzer der Ressource, der einer Anwendung erlauben möchte, in seinem Namen zu handeln. In der Regel ist dies eine Person.
- ***Protected resource**:* Die geschützte Ressource, auf die der Besitzer Zugriff hat. Diese kann verschiedene Formen haben, in der Regel handelt es sich jedoch um eine Web-API, die verschiedene Zugriffsrechte haben kann (Lesen, Schreiben …)
- ***Client**:* Der Client ist die Anwendung, die im Namen des Besitzers der geschützten Ressource auf diese Ressource zugreift. Diese Anwendung kann eine Anwendung auf einem Server sein, eine Javascript-Anwendung oder eine systemeigene Anwendung.
- ***Authorization server**:* Der Autorisierungsserver vergibt ein Access Token, das der Client verwenden kann, um anstelle des Besitzers der Ressource zu handeln. Hierzu authentifiziert er den Besitzer und überprüft seine Zustimmung.

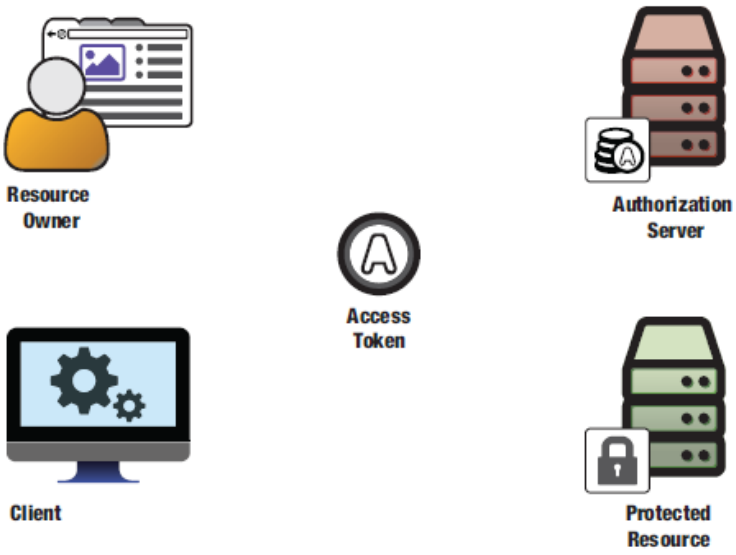


Abbildung 1 Die Rollen bei OAuth 2.0

1.2. Etappen des Autorisierungsprozesses

Bei Verwendung des Ablaufs client credential handelt eine Client-Anwendung nicht im Namen eines Dritten, sondern in ihrem eigenen Namen. In diesem Szenario ist demnach der Client der Resource Owner.

1.2.1. Identifizierung und Authentifizierung des Client

Zu Beginn des Prozesses kontaktiert der Resource owner den Client (1) und teilt ihm mit, dass er möchte, dass dieser in seinem Namen handelt. Auf diese Anfrage hin kontaktiert der Client den Authorization server (2) und gibt an, dass er anstelle des Ressource owner handeln möchte. Hierzu bringt der Client den Resource owner mit dem Authorization server in Kontakt, indem er verschiedene Informationen (3) übermittelt, die es ermöglichen, die angeforderten Berechtigungen für die Ressource zu identifizieren (über den scope), sowie eine URL, die es dem Authorization server erlaubt, den Client zu kontaktieren, sobald die Anfrage genehmigt wurde.

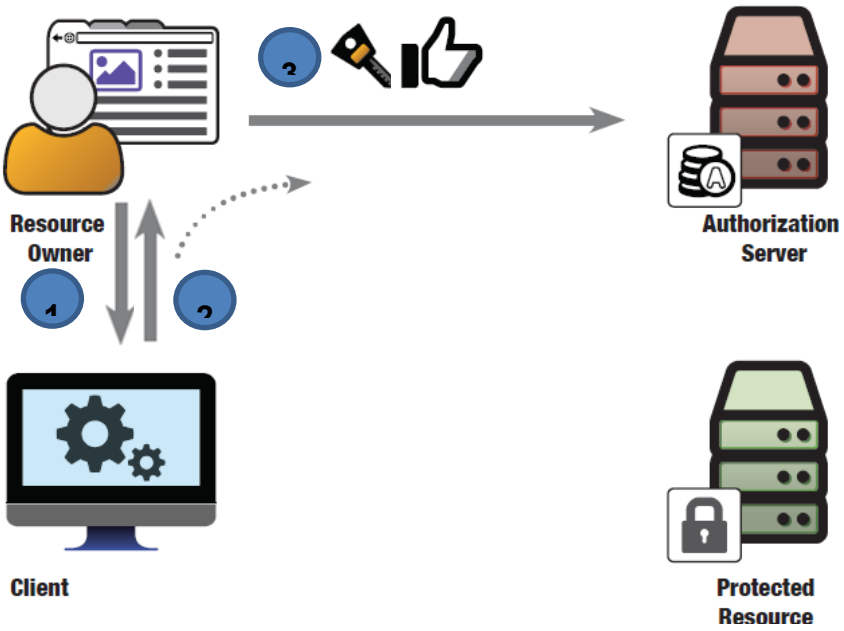


Abbildung 2 Der Client leitet den Resource owner zur Authentifizierung und Genehmigung an den Authorization server weiter

1.2.2. Token-Anfrage bei Autorisierungsserver

Ist der Ressource ownerbeim Authorization serveridentifiziert und authentifiziert, fordert dieser die Zustimmung des Resource owner (1) unter Angabe der vom Client angeforderten Berechtigungen (scope) für die Ressource an. Hat der Authorization server die Zustimmung erhalten, leitet er den Resource owner zum Client (2) weiter, indem er die zuvor erhaltene Weiterleitungs-URL verwendet.

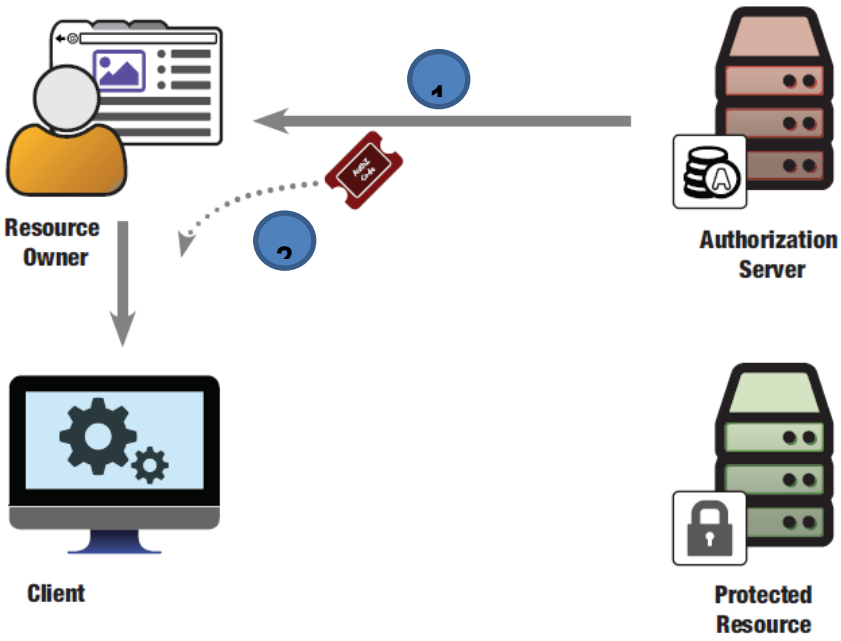


Abbildung 3 Der Authorization server übermittelt einen Authorization Code an den Client

1.2.3. Zugriff auf eine geschützte Ressource

Verfügt der Client über ein Access Token, kann er dieses Access Token, solange es gültig ist, verwenden, um auf die Ressource zuzugreifen. Hierzu übermittelt er seine ID, seinen Schlüssel und das Access Token an den Resource server(1). Dieser kontaktiert den Authorization server durch Übermittlung dieser Daten (2), damit dieser die Legitimität der Anforderung überprüft. Wurde die Anforderung durch den Authorization server genehmigt (3), sendet der Resource server dem Client die angeforderte Ressource.

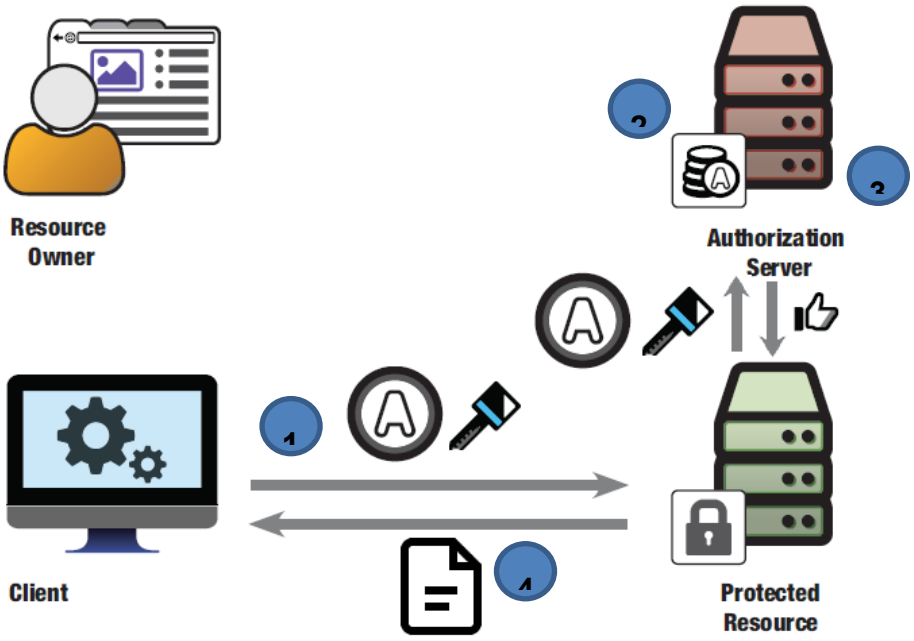
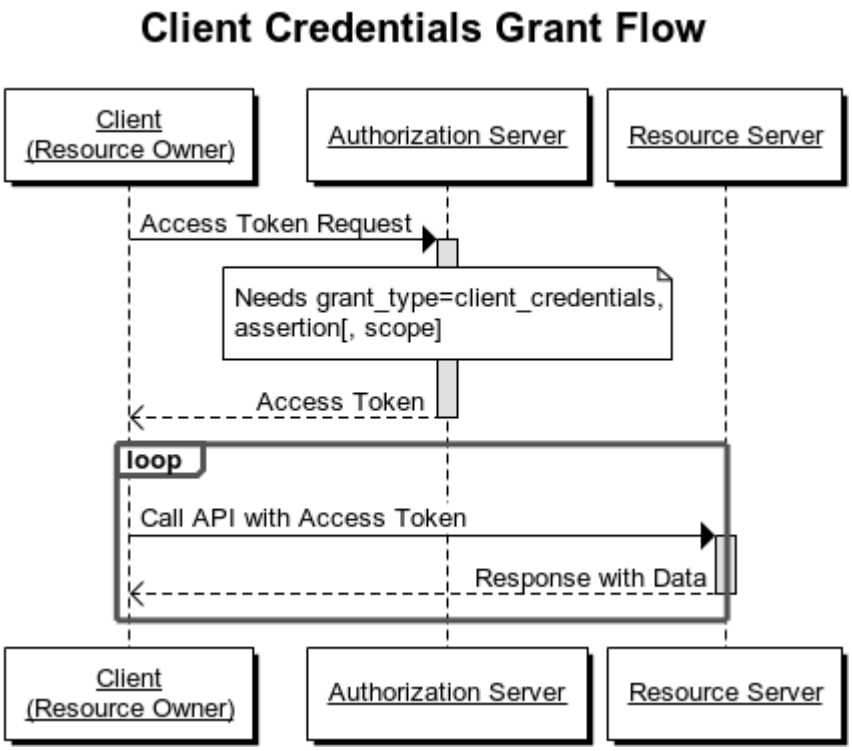


Abbildung 4 Verwendung eines Access Token, um auf eine mit OAuth geschützte Ressource zuzugreifen

2. Integration der Client Credentials

Dieser Abschnitt behandelt den Genehmigungsablauf client credential für der Dimona REST Web Service. Dieser Ablauf wird verwendet, wenn der Client in seinem eigenen Namen handelt. Dieser Ablauf dient eher der Authentifizierung juristischer Personen als natürlicher Personen. Dieser Ablauf wird deshalb von Unternehmen oder Diensteanbietern verwendet, die in ihrem eigenen Namen auf eineREST-Ressource zugreifen möchten.

In diesem Abschnitt finden Sie alle nötigen Informationen, um Ihre Anforderung an den Autorisierungsserver zu stellen, sowie die möglichen Antwortformate.



2.1. Client Credentials grant flow

Der Client kann ein access token nur durch Angabe seiner credentials (oder einer anderen unterstützten Authentifizierungsart) anfordern, wenn der Client Zugriff auf eine Ressource unter seiner Kontrolle anfordert oder wenn er Zugriff auf geschützte Ressourcen auf Grundlage einer vorher vom Autorisierungsserver erteilten Autorisierung anfordert.

Der Genehmigungsablauf client credential darf nur von geheimen Clients verwendet werden.

2.1.1. Autorisierungsanforderung und Antwort

Da die Authentifizierung des Clients als Gewährung der Autorisierung verwendet wird, ist keine weitere Autorisierungsanforderung erforderlich.

2.1.2. Access token request

Der Client stellt eine Anforderung an den token endpoint, indem er folgende Parameter im Format „application/x-www-form-urlencoded“ mit UTF-8-Codierung in die HTTP-Anforderung einfügt. Diese Parameter stammen direkt aus RFC 7519:

grant_type

ERFORDERLICH. Der Wert muss lauten „client_credentials“.

scope

OPTIONAL. Der Scope der Anforderung.

client_assertion_type

ERFORDERLICH. Der Wert muss lauten „urn:ietf:params:oauth:client-assertion-type:jwt-bearer“

client_assertion

ERFORDERLICH. Ein signiertes JWT, das den Client beim Autorisierungsserver authentifiziert wie in RFC 7519 beschrieben. Der Inhalt des JWT ist im Abschnitt „Client authentification“ beschrieben.

Hier ein Beispiel für eine HTTP-Anforderung:

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

Ist die Anforderung eines access token gültig und autorisiert, vergibt der Autorisierungsserver ein access token. Ein refresh token wird nie im Genehmigungsablauf client credential vergeben. Schlägt die Anforderung fehl oder ist sie ungültig, antwortet der Autorisierungsserver mit einer Fehlermeldung.

Die Antwort enthält die folgenden Parameter aus RFC 7519:

access_token

Der vom Autorisierungsserver vergebene access token.

token_type

Typ des ausgegebenen Tokens wie in RFC6749 section 7.1 beschrieben.

expires_in

Die Lebensdauer des access token in Sekunden. Der Wert „3600“ zum Beispiel gibt an, dass das access token eine Stunde nach Generierung der Antwort abläuft.

Beispiel für eine korrekte Antwort:

```
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

Der Autorisierungsserver antwortet mit dem HTTP-Antwortstatuscode 400 (fehlerhafte Anforderung) und gibt die folgenden Parameter aus RCF 7519 an:

error

ERFORDERLICH. Einer der folgenden eindeutigen ASCII-Fehlercodes [USAASCII]:

invalid_request

In der Anforderung fehlt ein notwendiger Parameter, für einen Parameter (außer dem grant type) ist ein ungültiger Wert angegeben, ein Parameter wurde wiederholt. Die Anforderung verwendet mehr als einen Mechanismus, um den Client zu authentifizieren, oder ist falsch aufgebaut.

invalid_client

Die Authentifizierung des Client ist fehlgeschlagen (Client unbekannt, Authentifizierung des Client nicht enthalten oder Authentifizierungsmethode nicht unterstützt).

invalid_grant

Der verwendete Authentifizierungsablauf ist fehlerhaft.

unauthorized_client

Der authentifizierte Client ist nicht autorisiert, diesen Ablauftyp zu verwenden.

unsupported_grant_type

Der Ablauf wird vom Autorisierungsserver nicht unterstützt.

invalid_scope

Der angeforderte Scope ist ungültig, unbekannt, fehlerhaft oder wird vom Ressourcenbesitzer nicht angeboten.

error_description

OPTIONAL. Meldung mit zusätzlichen Informationen zum aufgetretenen Fehler.

error_uri

OPTIONAL. Eine URL zu einer Webseite mit einer Dokumentation zum aufgetretenen Fehler.

Beispiel für eine falsche Antwort:

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

Ein signiertes JWT, das den Client im Autorisierungsserver authentifiziert, wie in RFC 7519 beschrieben. Der Inhalt des JWT ist im Abschnitt „Client authentification“ im RFC beschrieben.

jti

ERFORDERLICH. Dieser Parameter liefert eine eindeutige ID für das JWT. Groß- und Kleinschreibung sind zu beachten.

iss

ERFORDERLICH. Dieser Parameter identifiziert die Einheit, die das JWT ausstellt. Groß- und Kleinschreibung sind zu beachten.

sub

ERFORDERLICH. Dieser Parameter identifiziert das Hauptsächliche, das Gegenstand des JWT ist. Groß- und Kleinschreibung sind zu beachten, ein String- oder URI-Wert ist enthalten.

aud

ERFORDERLICH. Dieser Parameter definiert den Empfänger des JWT.

exp

ERFORDERLICH. Dieser Parameter definiert das Ablaufdatum ab dem oder nach dem das JWT nicht mehr akzeptiert werden darf.

nbf

ERFORDERLICH. Dieser Parameter identifiziert den timestamp, den Zeitpunkt vor dem das JWT nicht akzeptiert werden darf.

iat

ERFORDERLICH. Dieser Parameter definiert den timestamp zu dem das JWT erstellt wurde.

3. URLs

Server-URL: **https://services.socialsecurity.be/**

Zugangspunkt:

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

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