Algemene Inleiding Open Authenticatie (OAuth)

Logius Praktijkrichtlijn
Werkversie 02 april 2025

Deze versie:
https://logius-standaarden.github.io/OAuth-Inleiding/

Laatst gepubliceerde versie:
https://gitdocumentatie.logius.nl/publicatie/api/oauth-inleiding/

Laatste werkversie:
https://logius-standaarden.github.io/OAuth-Inleiding/

Redacteur:
Logius Standaarden (Logius)

Auteur:
Martin van der Plas (Logius)

Doe mee:
GitHub Logius-standaarden/OAuth-Inleiding
Dien een melding in
Revisiehistorie
Pull requests

Dit document is ook beschikbaar in dit niet-normatieve formaat: pdf

Samenvatting

Dit document geeft een inleiding op Open Authentication, kortweg OAuth. De laatste werkversie van de OAuth.NLGov standaard is gepubliceerd op github.io, de laatste vastgestelde versie is gepubliceerd op Logius.nl en de bronbestanden staan in de github repositories van Logius standaarden die zijn gemarkeerd met het topic "oauth"

Status van dit document

Dit is een werkversie die op elk moment kan worden gewijzigd, verwijderd of vervangen door andere documenten. Het is geen door het TO goedgekeurde consultatieversie.

Inhoudsopgave

1. Inleiding document

Dit document biedt een introductie in de OAuth standaard en met name het OAuth NLGov profiel wat op op de lijst verplichte standaarden is opgenomen van het Bureau Forum Standaardisatie link

Het begrijpen en doorgronden van de implicaties van de OAuth standaarden en de profielen daarop blijken een steile leercurve te hebben en de technisch inhoudelijke documenten zijn niet geschikt voor een brede doelgroep die geen specialistische ICT kennis bezit van APIs, Authorisatie en security. Vandaar dit document waarmee we een algemeen beeld willen schetsen van de standaard. Dit doen we aan de hand van een praktijkvoorbeeld, een samenvatting van het profiel, een schets van de context en architectuur en verschillende use cases. Het document is zo opgebouwd dat alle hoofdstukken afzonderlijk kunne nwordn gelezen en voor verdieping wordt veelvuldig verwezen naar de standaard. Ook wordt er vanuit de standaard weer verwezen naar dit document voor meer algemene context.

2. OAuth - De basis

2.1 Inleiding

Onlangs kregen we de vraag of er meer basisinformatie is over OAuth 2.0. Een heel begrijpelijke vraag. De huidige standaard die door het forum standaardisatie op de lijst van verplichte standaarden is gezet vereist namelijk deze basiskennis om te begrijpen wat de standaard nu eigenlijk verplicht stelt.

Een deel van het antwoord zit m al in de formele naamgeving van de standaard. Het betreft namelijk een profiel ofwel een vastgestelde configuratie flow voor de Nederlandse overheden op de formele OAuth 2.0 standaard.

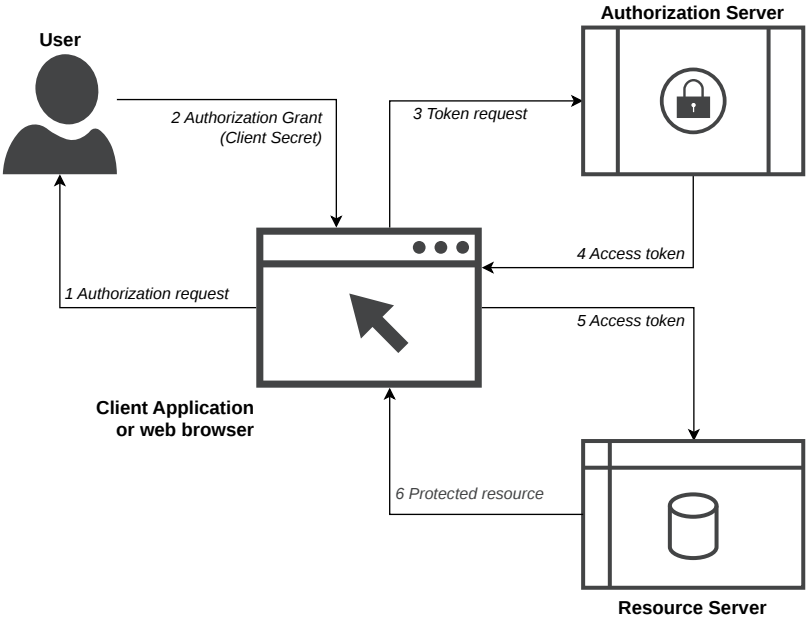
2.2 Context

OAuth 2.0 is een authorizatieframework wat het mogelijk maakt om gecontroleerde toegang te krijgen tot gebruikersaccounts op een HTTP service zoals bijvoorbeeld Google, Facebook, Spotify etc. De standaard werkt op basis van het delegeren van de user authentication aan de service die het user account host en door applicaties van derden te autoriseren om het user account te hergebruiken. Hierdoor kunnen gebruikersrechten of -gegevens met een website of applicatie gedeeld worden zonder wachtwoorden te delen.

Deze interactie wordt altijd beschreven in een flow. Het Nederlandse profiel beschrijft alle aspecten van de door ons gewenste flow(s). Momenteel is alleen de Authorization Code Flow onderdeel van het Nederlandse profiel. Later meer hierover. Een aantal aspecten zijn wel essentieel en dus ook randvoorwaardelijk voor een gebruikelijke autorisatie op basis van OAuth 2.0:

1. een *user* met een *account* : bijvoorbeeld example@logius.nl
2. een *authorization service* : waar de user zich kan identificeren en authenticeren : bijvoorbeeld Facebook/Apple/Google
3. een *resource service* : waar de inhoudelijke vraag aan wordt gesteld en waar de data/resources zijn opgeslagen : bijvoorbeeld Spotify
4. een *client applicatie* : waarmee de user z'n account gegevens invult en de vraag stelt aan de authorization service en de resource service : bijvoorbeeld de Spotify app of een test tool als Postman.

§ **2.2.1 In onderstaand schema zijn de vier bovenstaande elementen en de onderlinge flow weergegeven:**

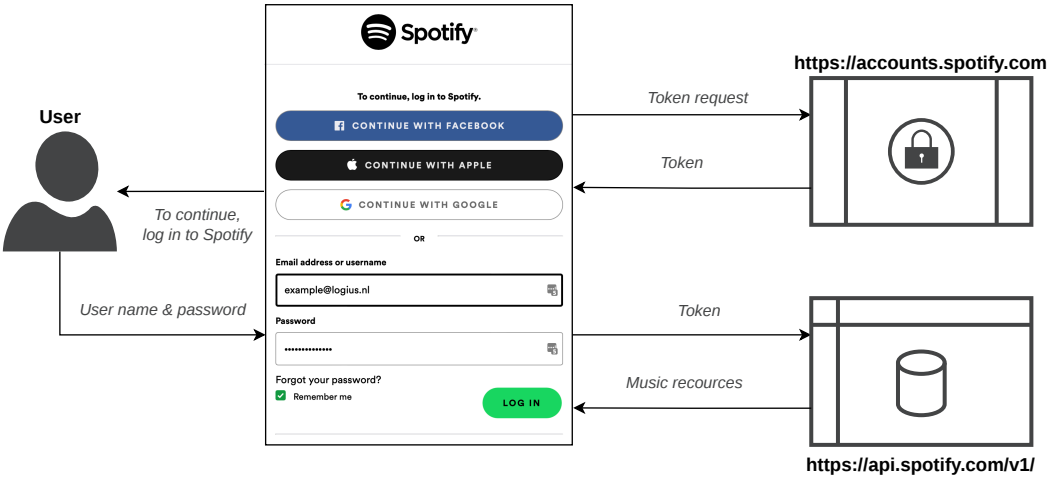


§ **2.2.2 De flow is dan als volgt:**

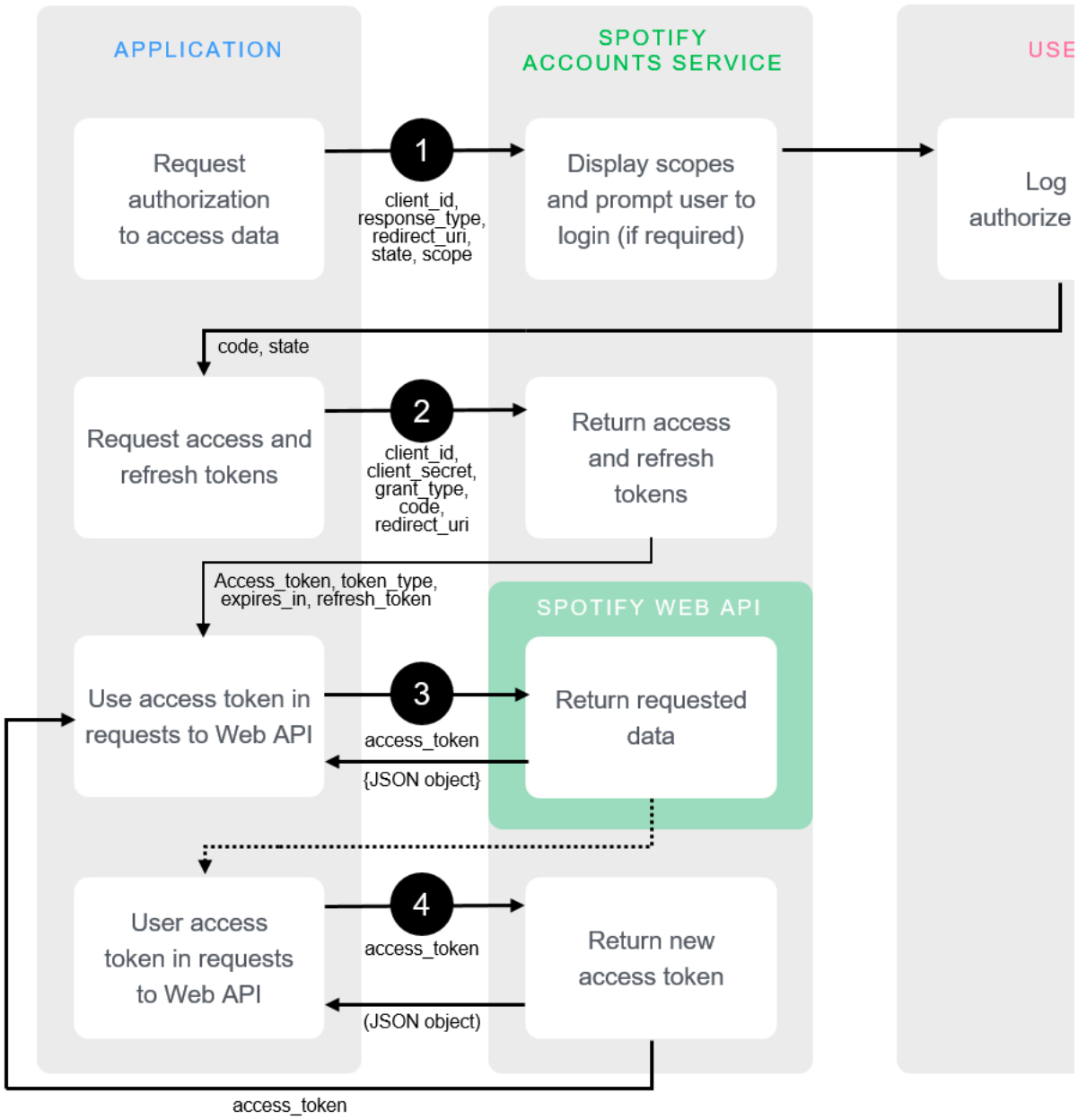
1. De **user** navigeert naar de client application of webbrowser (***de client***) en wordt gevraagd in te loggen (het *authorization request*).
2. De **user** voert z'n account gegevens in die alleen hij weet (het *client secret*). Uiteraard doet de user dit alleen als hij de client applicatie vertrouwd.
3. De **client** stuurt op basis van de gegevens van de **user** een verzoek aan de ***authorization server*** (het *token request*). De **authorization server** beoordeeld dit verzoek op basis van alle beschikbare **user** en **client** gegevens. (Veelal dient een client applicatie zich vooraf eerst te registreren.)
4. De **authorization server** stuurt als alles goed is een *Access token* terug aan de **client**. Dit is gebruikelijk een JSON Web Token (JWT).
5. De **client** stuurt namens de **user** een request aan de resource server en voorziet dit request van het *access token*. De resource server kan de combinatie van het request en de en token beoordelen en bepalen welk response aan de client wordt gestuurd.
6. Indien alles aan de vereisten voldoet stuurt de **resource server** een response aan de **client** met de gevraagde data (de *protected resource*)

§ 2.3 Voorbeeld

Zoals al aangegeven in de context werkt een voorbeeld het beste. In onderstaand schema is het voorbeeld opgenomen van Spotify waarbij een user, in dit geval example@logius.nl, inlogt op de Spotify webclient. Het voorbeeld gebruikt de authorization server van Spotify zelf om een token te verkrijgen en daarna kan de user z'n persoonlijke gegevens, afspeellijsten en muziek opvragen bij de Spotify API.



Zowel de webclient van Spotify als de client applicatie of app gebruiken dezelfde API om resources op te vragen. De "Coursera basis training met Postman" die in de Referenties wordt genoemd legt uit hoe je deze flow kan naspelen met als client de testtooling van Postman. De Coursera training is met name interessant om verdere kennis op te doen. De training is gebaseerd op Postman als client en gebruikt Spotify als server om tegen aan te praten. Hiervoor log je in op https://developer.spotify.com/ en maak je een App aan in het dashboard. In het voorbeeld is Spotify zowel de Authorization Server als de Resource server. Spotify beschrijft in het voorbeeld en de documentatie helder hoe de Authorization Code Flow precies werkt (zie https://developer.spotify.com/documentation/general/guides/authorization/code-flow/) en dit is ook precies de flow die in het NL Profiel wordt gebruikt. Het gedetailleerde schema wat Spotify gebruikt om de flow toe te lichten is als volgt:



§ 2.4 Architectuur

De kern van OAuth is uiteraard het scheiden van de Authorization Server van de Resource Server en deze onafhankelijk te maken van de gebruikte client. Dit blijkt mooi uit bovenstaande flow en voorbeeld. Belangrijkste implicatie voor de architectuur is daarmee dan ook dat voor een dergelijke oplossing waarbij OAuth wordt toegepast de user niet alleen een client en een resource server wordt aangeboden, maar ook een authorization server (drie autonome architectural building blocks). Dit kan een authorization server zijn van de organisatie zelf, zoals in het voorbeeld, maar ook een authorization server van een derde partij zoals in de context al wordt gesuggereerd en zoals je kan zien in het inlogscherm van Spotify waarbij je ook kan registreren met Facebook, Apple of Google. In de context van de Nederlandse overheidsarchitectuur is het dus van belang bij een solution architectuur voor een voorziening goed na te gaan en documenteren welke partijen worden voorzien in de genoemde building blocks. Zie ook het theme IAM en API van de Nora en uiteraard de genoemde standaarden zoals gepubliceerd door Logius en het Forum Standaardisatie.

§ 2.5 Referenties

[Coursera basis training met Postman]

https://www.coursera.org/projects/api-testing-a-real-application-via-postman

[Link naar de lijst van verplichte standaarden]

https://forumstandaardisatie.nl/open-standaarden/verplicht

[link naar de standaard]

https://forumstandaardisatie.nl/open-standaarden/verplicht#:~:text=NL%20GOV%20Assurance%20profile%20for%20OAuth%202.0

[link naar de logius standaard]

https://gitdocumentatie.logius.nl/publicatie/api/oauth/

[link naar het IAM thema van de NORA]

https://www.noraonline.nl/wiki/Identity_%26_Access_Management_(IAM)

[link naar het API Thema van de NORA]

https://www.noraonline.nl/wiki/API

[Het JWT token kan men eenvoudig decoden/inspecteren op]

http://jwt.io

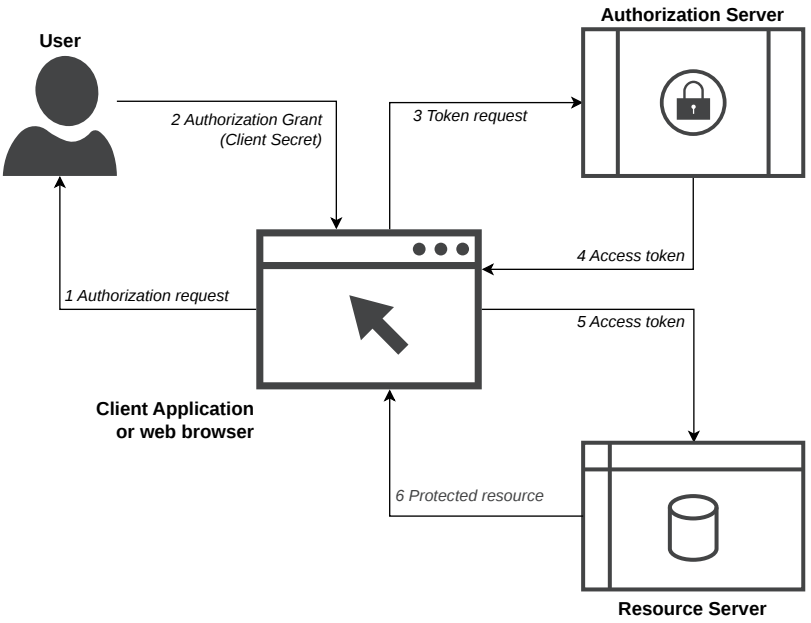
3. OAuth - Samenvatting van de OAuth-NL standaard

> link naar de standaard: https://gitdocumentatie.logius.nl/publicatie/api/oauth/v1.0 Versie 1.0 vastgesteld op 09 juli 2020

3.1 Inleiding

Deze standaard is voor het ontsluiten van services van publieke/overheidsorganisaties. De services ontsluiten data via een zogenaamde Resource Server (verder aangeduid als de API). De resource server ofwel de API is afhankelijk van een Authorization Server. Deze Authorization Server voorziet in de identificatie en autorisatie van de user die de API bevraagd. De user gebruikt een Client die hij dient te vertrouwen om namens hem de vraag naar de API te sturen en de response op de vraag te ontvangen en presenteren.

3.1.1 Schematische weergave van de flow van de standaard



§ 3.2 Vereisten

De standaard biedt simpel gezegd access control voor RESTful API's. Dergelijke API's worden momenteel door de overheid ontwikkeld conform de API Design Rules (ADR) en beschreven conform de Open API Specification (OAS).

Deze standaard beschrijft vereisten voor de API (Resource Server), Authorization Server en de Client. Hieronder is een samenvatting van de vereisten toegelicht:

§ 3.2.1 1. Introductie

- Iedere Authorization Server moet voldoen aan alle vereisten uit de standaard (§ 1.3)
- Iedere Client moet alle functies uit de standaard gebruiken (§ 1.3)
- Iedere API moet alle functies uit de standaard gebruiken (§ 1.3)

§ 3.2.2 2. Client profiel

- Clients zijn *Full* clients, ofwel web applicaties die centraal draaien, of *Native* clients, instanties van software die draaien op het device van de user, de zogenaamde apps. Beide client types hebben verschillende vereisten (§ 2.1)
- Clients moeten vooraf zijn geregistreerd bij de Authorization Server (§ 2.2)
- Clients mogen geen gebruik maken van een redirect naar de localhost en mogen ook geen waardes doorsturen naar andere URI's (§ 2.2.1)
- Clients moeten een willekeurige status parameter genereren en koppelen aan de client sessie om deze vervolgens mee te sturen naar de Authorization server en te verifiëren of deze ook correct in de response wordt meegegeven (§ 2.3)
- Clients moeten de volledige redirect URI meesturen in het verzoek aan de Authorization server (§ 2.3)
- De Authorization Server moet de redirect URI controleren ten opzichte van de URI die vooraf is geregistreerd door de Client (§ 2.3)
- Native Clients moeten PCKE toepassen (§ 2.3)
- Wanneer de API, Client en Authorisation Server niet onder verantwoordelijkheid vallen van één organisatie moeten PKIOverheid certificaten worden gebruikt (met OIN) (§ 2.3.4)
- Clients moeten autorisatie requests over TLS sturen en moeten het certificaat van de API verifiëren (§ 2.4)

§ 3.2.3 3. Authorization Server profiel

- De Authorization Server moet alle communicatie versleutelen met TLS en voldoen aan alle security eisen (§ 3)
- De Authorization Server moet dynamic client registration toestaan. De autorisatie server mag de Scopes beperken van dynamisch geregistreerde clients (§ 3.1.3)
- De Authorization Server moet de user laten weten welke Client is geregistreerd en welke access die Client vraagt (§ 3.1.4)
- De Authorization Server moet OpenID.Discovery aanbieden (§ 3.1.5)
- De Authorization Server moet op verzoek van de Client tokens intrekken (§ 3.1.6)
- De Authorization Server moet JWT tokens verstrekken die de API kan verifiëren (§ 3.2.1)
- De Authorization Server moet authenticatie vereisen voor de revocation en introspection endpoints (§ 3.2.2)
- Alle uitgegeven tokens mogen worden ingetrokken (§ 3.4)
- Access tokens hebben verschillende lifetimes (§ 3.4)
- De Authorization Server zou Scopes moeten definiëren en documenteren (§ 3.5)

3.2.4 4. Protected Resource (API) profiel

- De API geeft de Client toegang wanneer deze een geldig access token en de correcte Scope heeft. De API vertrouwd erop dat de Authorization Server de security en access control borgt (§ 4.1)
- De API (met vertrouwelijke data) die een hoger vertrouwensniveau vereist van de eindgebruiker moet de data alleen beschikbaar stellen binnen een unieke Scope (§ 4.1)
- De Client die vertrouwelijke data wil opvragen bij de API moet een hoger vertrouwensniveau Scope meegeven in het verzoek aan de Authorization Server (§ 4.1)
- De Authorization Server moet de authenticatie van de eindgebruiker op het juiste vertrouwensniveau vaststellen (§ 4.1)
- Een API moet Bearer tokens accepteren in de authorization header en mag deze ook accepteren als form parameter (§ 4.2)
- Een API mag geen access tokens accepteren als query parameter (§ 4.2)
- Een API moet documenteren welke scopes vereist zijn voor toegang tot vertrouwelijke data (§ 4.2)
- Een API moet een access token verifiëren (§ 4.3)
- Een API moet limiteren welke Authorization Servers het vertrouwt (§ 4.3)

3.2.5 6. Security overwegingen

- Alle transacties moeten worden versleuteld met TLS en het is aanbevolen om hierbij de richtlijnen van het NCSC op te volgen (§ 6)

3.3 !! Disclaimer

Deze samenvatting is een interpretatie van de standaard en is niet volledig in de opsomming of beschrijving van alle vereisten. Wanneer de standaard moet worden geïmplementeerd dient altijd de officiële en actuele standaard te worden gebruikt zoals gepubliceerd door het Forum Standaardisatie op: https://forumstandaardisatie.nl/open-standaarden/nl-gov-assurance-profile-oauth-20.

4. Use Cases

4.1 Use Case Digital System Environmental Law

4.1.1 Overview

In this use case, the client credential flow is used in a digital system that enables citizens to request permits for activities they want to undertake in their environment. For example when a citizen wants to cut down a tree or change the exterior of their house they have to request a permit. The citizen can start a permit request in the web interface of the Digitaal Stelsel Omgevingswet - Digital System Environmental Law (hereafter: DSO) and this system will determine what local authorized authority (e.g. a municipality) is responsible. Once the permit request is ready, a trigger is sent to the system of the local authorized authority to use the client credential flow to retrieve the data associated with the request. The request can then be completed in the client software of that local authority.

4.1.2 Resource owner

The resource owner is the citizen that provided the information during the permit request.

4.1.3 Client

In this use case the client is the software that the authorized authority (e.g. the local authority, municipality, county etc.) who wants to retrieve information from the digital system, uses.

4.1.4 Authorization server

The authorization server is owned by the DSO, it is part of their API Management platform.

4.1.5 Resource server

The resource server is the backend server of the DSO where the permit request is available.

4.1.6 Scopes & claims

The scopes and claims used in this system are defined by the DSO, and are related to the CRUD actions of the API's available.

4.1.7 Rationale

The DSO exposes API's secured on different levels. To standardise as much as possible, OAuth is used in several of the authentication and authorization flows. Therefore besides the Authorization code flow where an enduser is involved, the client credentials flow is used between systems in order to use the same authorization mechanism for as many usecases as possible.

4.2 Rights delegation key registry Buildings & adresses

4.2.1 Overview

Key registries in Dutch government often have a single registry where all data is collected but multiple source holders are responsible for maintaining parts of the data. Source holders, such as municipalities, should not be allowed to modify each others data without permission. It is very common to either have commercial cloud providers or partnerships of cooperating municipalities maintain a key registry on behalf of multiple sourceholders. Currently Dutch Cadastre hosts multiple key registries where this is the case each with its own application level solution to allow for this. Within the key registry adresses and buildings a PoC has been executed to show how OAuth can solve this in a generic way. This has the advantage of cutting down on custom code and affering a uniform procedure to source holders who often maintain multiple key registries at Dutch Cadastre.

4.2.2 Resource owner

A municipality that is legally required to maintain information on its buildings and adresses.

4.2.3 Client

A cloud provider or partnership of cooperating municipalities hosts an application that operates on behalf of the legal entity (municipality) to update and maintain the key registry Buildings & Adresses. This application acts as client that connects to the API of the Dutch Cadastre.

4.2.4 Authorization server

manager The authorization server is hosted by Dutch Cadastre and provides (among other things) authorizations to municipalities to update and maintain data on buildings and adresses.

4.2.5 Resource server

The central server that hosts the key registry for buildings and adresses at dutch Cadastre.

4.2.6 Scopes & claims

A JWT token containting the following claim is used: The claim "cdi" contains the ID of the municipality delegating its rights to maintain the key registry. The delegatee, the cloud provider gets a token containing this claim from the authorization server when providing its credentials. The actual delegation, provding client credentials to the client and delegating rights from the resource owner to the cleint is done out of band as this is not part of the client credentials flow.

4.2.7 Rationale

The client credentials flow is used instead of the authorization codeflow because the resource owner is a legal entity and the client acts on behalf of this legal entity, not the individual user working with the application.

4.3 Overview

The Digitale Delta API provides access to measurements and related information. Consumers of the data, in general, will be automated processes or systems such as Business Intelligence or GIS systems. Since not all data may be open or public, access to certain data must be restricted.

For this scenario, an interactive authentication method is not feasible and not all devices may equiped with PKI Overheid certificates.

Next to requesting data, the API allows for adding or removing data by using import files or sensor devices.

Client credential flow does offer the required functionality.

4.4 Rights delegation Air sensors network

4.4.1 Overview

With the deployment of an endless number of air sensors the quality of the gathered data depends on the security of the sensor network. In this usecase the sensors are connected to a LoRaWAN network and equiped with a unique ID (GUID) for authentication. This prevents the injection of unindentifiable data into the network.

4.4.2 Resource owner

The resource owner is the sensor owner. In this case the the sensors are provided to citizens by the Resource owner, e.g. a local government or city.

4.4.3 Client

The Air sensor itself functions as a client, uploading each measurement to the API of the Resource owner via the LoRaWAN network.

4.4.4 Authorization server

In this use case the resource owner also provides the Authorization server. The unique id's of the sensors are preloaded in the authorization server.

4.4.5 Resource server

The resource server is provided by the sensor provider, e.g. a local government or city, the resource server provides the API where all sensors write their data.

4.4.6 Scopes & claims

not applicable

4.4.7 Rationale

To mitigate the risk of incorrect data or manipulation of sensor data the sensor will connect to the authorization server first while onboarding (Step 1). The authorization server checks the provided sensor ID and provides the sensor with a token (Step 2). The sensor then starts to collect data and upload the data to the provided API and includes the token with each request to the resource server.

5. Detail onderwerpen OAuth

5.1 OAuth details

Verdieping op Par 3.1.1 & Token exchange

5.1.1 Token exchange

Where possible, the token exchange [*rfc8693*] grant type SHOULD be implemented instead of client credentials grant type, as this proves the identity of the user (and, where applicable, a second user that may act on behalf of the user).

To Do add as a third flow in this document in usecases

Voorbeelden token exchange (rfc8693)

• Impersonation. Een achterliggende applicatie doet namens een andere applicatie een API aanroep met een tweede token (delegation scenario, met act claim). Het tweede token zal vaak minder of andere scopes of audience restricties hebben dan het originele token. Een ander bekend voorbeeld is dat het token slechts geldig is in de context van één transactie, en/of dat het token langer geldig is, bijvoorbeeld bij asynchrone (batch) verwerking van gegevens.

• Delegation. Een gebruiker (vertegenwoordigd door een actor token) acteert namens een andere gebruiker (subject token).

@@@ [*rfc8693*] distinguishes impersonation and delegation scenarios.

5.2 OIN en PKIO in relatie tot OAuth NLGov profiel

> Concept beschrijving dd 22-08-2024 - door Martin van der Plas

5.2.1 Inleiding

De nieuwe versie van het OAuth NL profiel voegt onder meer het Client credentials profiel toe. Zie ook v1.1.0-rc.2 op : https://logius-standaarden.github.io/OAuth-NL-profiel/

Op basis van deze toevoeging kwamen, na vaststelling door de werkgroep en de publieke consultatie, aanvullende vragen over het gebruik van PKIO certificaten en het OIN vanuit het onderwijs domein. De Edukoppeling werkgroep gebruikt voor het definiëren van een aantal OAuth best practices[1] ook het NL GOV Assurance profile for OAuth 2.0 (NL GOV).

Aangezien het NLGov OAuth profiel is opgesteld op basis van het iGov profiel is het soms wat lastig om te doorgronden wat de precieze vereisten zijn. Ook is de materie inzake oauth relatief complex en wordt er nogal eens wat impliciete voorkennis verondersteld. Dit atikel dient dan ook niet als vervanging van de standaard of als aanvulling daarop maar meer als richtlijn of toelichting.

5.2.2 Context:

OIN refenties De standaard heeft de volgende relevante verwijzingen naar het OIN die hieronder verder worden toegelicht

- Par 2.3.4 https://logius-standaarden.github.io/OAuth-NL-profiel/#client-keys beschrijft de connectie van de client met de Authorization server en vereist simpel gezegd dat clients een keypair moeten hebben om zich te authenticeren bij het token endpoint. in onze additionele content eisen we dat hiervoor een PKIO certificaat met OIN moet worden gebruikt als de authorization server, resource server en client niet allemaal van dezelfde organisatie zijn. Aanvullend eisen we dat clients hun public key moeten registreren in de client metadata
- Par 3.2 https://logius-standaarden.github.io/OAuth-NL-profiel/#connections-with-protected-resources beschrijft hoe resources connecten met de authorization server. het vereist dat de authorization server JWT tokens uitgeeft. Ook hier eisen we dat hiervoor een PKIO certificaat met OIN moet worden gebruikt als de authorization server, resource server en client niet allemaal van dezelfde organisatie zijn.

5.2.3 PKI referenties De standaard heeft de volgende relevante verwijzingen naar PKI die hieronder verder worden toegelicht:

5.2.4 Intenties:

- Clients in de client credentials flow zijn pre-registerd Step 1. Client Authentication
- Private_key_jwt is naast tls_client_auth toegestaan. In beide situaties is een Private key nodig om ofwel de JWT te signen ofwel de mTLS connectie op te zetten. Indien dus de PKIO verplicht is is in beide gevallen een OIN inzichtelijk bij de authorization server.
- verbindingen tussen client, resource en authorization server zijn altijd versleuteld met TLS en indien PKIO verplicht is dus ook altijd the identificeren (op TLS niveau met de OIN uit het subjectSerialNumber veld van de public key)
-

5.2.5 Conclusie:

- een kwaadwillende kan nooit met een self signed certificaat een client faken die met een ander certificaat is geregistreerd bij de authorization server

§ **5.2.6 Achtergrondinformatie:**

§ *5.2.6.1 Vragen DUO:*

De stapjes binnen Edustandaard over het toepassen van OAUTH gaan langzamer dan ik zou willen, maar zeker wel de goede kant op. Ondertussen heb ik ook een nieuwe 'proposed' versie van NL-GOV (10 juli 2024) gezien. (Terzijde, er is ook een versie van 13 mei vindbaar, zonder dat duidelijk is dat die is vervangen, dat kan verwarrend zijn). Ook in deze nieuwe versie zit iets of juist niet, wat DUO denkt nodig hebben. Vandaar dat ik er een vraag over stel.

Eén van de requirements is redelijk scherp, "de identiteit (OIN) van de client wordt onomstotelijk vastgesteld".

In deze versie lees ik dit:

2.3.3 In addition to private_key_jwt, the client authentication method tls_client_auth [rfc8705https://logius-standaarden.github.io/OAuth-NL-profiel/#bib-rfc8705] MAY also be used. 2.3.4 Clients using the authorization code grant type or direct access clients using the client credentials grant type MUST have a public and private key pair for use in authentication to the token endpoint. These clients MUST register their public keys in their client registration metadata by either sending the public key directly in the jwks field or by registering a jwks_uri that MUST be reachable by the authorization server 2.3.4 In case the Authorization Server, Resource Server and client are not operated under responsibility of the same organisation, each party MUST use PKIoverheid certificates with OIN.

Hieruit spreekt dezelfde intentie uit als onze requirement. Dat is goed. Maar wordt het waar gemaakt? Het idee van DUO:

- Bij mTLS wel. - De server leest de metadata van het PKI certificaat uit. Haalt daaruit het root - certificaat (NL overheid) en ook het OIN (het SSN-veld) van de certificaat-houder.
- Bij private_key_jwt niet. - De publieke sleutel wordt uitgelezen met het jwks- of jwks_uri-veld. Deze versie van NL-GOV beschrijft een JSON Web Key Set (JWK set) van metagegevens dat daarmee toegankelijk is. Dus wel de publieke sleutel, maar niet het root-certificaat en het SSN-veld met OIN. Een kwaadwillende kan met een self signed certificaat een legitieme client faken.

Als dat inderdaad zo is, dan is private-key-jwt ongeschikt voor client credentials en hoort het niet thuis in dit deel van deze standaard.

✧ ✧ ✧

§ *5.2.6.2 Interpretatie Kennisnet:*

Mijn interpretatie bij de toepassing van een aantal NL GOV aspecten is als volgt:

Toepassing private_key_jwt voor client authenticatie en PKIo certificaat voor ondertekening:

1. De ondertekening is de jwt. De bij registratie opgenomen client_id en public key van client maken het mogelijk voor AS om ondertekening te verifiëren[2].

- Hiermee wordt bewezen dat client over private key PKIo beschikt (proof-of-possession authenticatie). Ik ga er hierbij vanuit dat client authenticatie van gelijkwaardig niveau is als mTLS. Tevens kan AS client_id aan OIN relateren (identificatie).

Toepassing PKIo (client en AS/RS niet onder beheer van dezelfde partij)

1. Omdat client niet door dezelfde partij wordt beheerd als AS/RS schrijven we conform NL GOV OAuth voor dat PKIo gebruikt moet worden.

- In combinatie met toepassing van private_key_jwt client authenticatie (en dus geen mTLS) betekent dit dus dat ondertekening van privat_key_jwt met PKIo certificaat wordt toegepast. 2. Het PKIo certificaat authenticeerd de rechtspersoon. Met het OIN in certificaat kan de rechtspersoon geïdentificeerd worden. Met het door de AS uitgegeven client_id kan een client (als onderdeel van het applicatielandschap van rechtspersoon) geïdentificeerd worden.

- Er is dus identificatie op 2 niveaus, OIN en client_id. NL GOV OAuth vereist niet dat de client_id het OIN is. Dit wordt echter niet expliciet aangegeven. Binnen het onderwijs biedt het voordelen op verschillende niveaus te kunnen identificeren, maar toepasbaarheid is onduidelijk. Ook onduidelijk of op dit niveau niet uniformiteit/standaardisatie wenselijk is.

3. Voor AS server geldt bij toepassing van private_key_jwt dus GEEN mTLS maar wel toepassing van PKIo voor serverbeveiliging (SSL/TLS) en bepaalde TLS versie.

- In NL GOV staat bij paragraaf 2.4.1 Requests to the Protected Resource: Authorized requests MUST be made over TLS, and clients MUST validate the protected resource server's certificate. Ik neem aan dat dit ook voor requests naar AS geldt als client authenticatie obv private_key_jwt wordt toegepast.
- In deze situatie wordt AS en RS serverbeveiliging vereist op basis van TLS en PKIo. Er gelden dan ook aanvullende eisen rond TLS versie en ciphert suites (NCSC/DK) die expliciet aangegeven moeten worden.

Discovery

1. NL GOV OAuth - Discovery (paragraaf 3.1.5) stelt :"The authorization server MUST provide an [OpenID Connect service discovery] [OpenID.Discovery] endpoint listing the components relevant to the OAuth protocol". \

- Er is onduidelijkheid wat de exacte scope / betekenis is van discovery. In het NL GOV OAuth profiel lijkt dit expliciet betrekking te hebben op de AS meta data. Het is onduidelijk of discovery ook betrekking heeft op meta data van client (het toepassen van een uri in plaats van opname in een jwks veld). Bij client authenticatie op basis van private_key_jwt kan zowel een jwks_uri gebruikt worden als een jwks veld. Is met het toepassen van discovery de toepassing van een jwks_uri logisch en de toepassing van private_key_jwt het jwks veld[3] onlogisch (en visa versa indien discovery juist niet vereist wordt zoals bij Edukoppeling OAuth Best Practices)?

✧ ✧ ✧

5.2.6.3 Links:

[1] https://www.edustandaard.nl/app/uploads/2024/07/2024-07-16-Edukoppeling-OAuth-Best-Practices-v1.0.pdf

[2] Een client moet zijn publieke sleutel vooraf registreren bij een autorisatieserver, zodat de server hiermee de ondertekening kan verifiëren.

[3] En hierin ook het PKIoverheid cert als x5c parameter (X. 509 Certificate Chain) en niet x5u. Dit bericht kan informatie bevatten die niet voor u is bestemd. Indien u niet de geadresseerde bent of dit bericht abusievelijk aan u is toegezonden, wordt u verzocht dat aan de afzender te melden en het bericht te verwijderen. De Staat aanvaardt geen aansprakelijkheid voor schade, van welke aard ook, die verband houdt met risico's verbonden aan het elektronisch verzenden van berichten. This message may contain information that is not intended for you. If you are not the addressee or if this message was sent to you by mistake, you are requested to inform the sender and delete the message. The State accepts no liability for damage of any kind resulting from the risks inherent in the electronic transmission of messages.

6. Orkestratie

6.1 Inleiding

API orkestratie is een cruciaal concept binnen de moderne ICT-architectuur, vooral in de context van het integreren van verschillende applicaties en services. Het stelt organisaties in staat om complexe workflows te beheren door verschillende API's te coördineren en hun interacties te stroomlijnen. Dit biedt voordelen zoals verhoogde efficiëntie, verbeterde dataconsistentie en een betere gebruikerservaring.

6.1.1 Wat is API Orkestratie?

API orkestratie verwijst naar het proces waarbij meerdere API-aanroepen worden gecoördineerd om een specifieke taak of workflow uit te voeren. Dit kan bijvoorbeeld inhouden dat gegevens van verschillende bronnen worden samengevoegd of dat meerdere services in een bepaalde volgorde worden aangeroepen om een einddoel te bereiken.

§ 6.1.2 Uitgangspunten

Uitgangspunt is de bestaande registraties met de bestaande registratie-processen.

Met betrekking tot security zijn er diverse aandachtspunten:

- Data integriteit. Dmv signing zou van individuele gegevens kunnen worden aangetoond wat de authentieke bron is en of dat deze identiek zijn aan de authentieke gegevens. Interessant vraagstuk is wat dit betekent als gegevens tijdens het orkestreren worden getransformeerd. Mogelijk raakvlakken met de RDF Dataset Canonicalization standaard.
- Authenticatie/autorisatie. Bij orkestratie worden verschillende requests uitgevoerd vanuit mogelijk verschillende identiteiten, verschillende identity stores en verschillende scopes/audiences. Hierbij kan zowel een transparant als niet-transparant model worden toegepast. Deze willen we beiden beschrijven. Relevante standaarden zijn FSC en OAuth Token Exchange.

§ 6.1.3 Use cases

Er zijn reeds enkele voorbeelde van uwe cases waarbij orkestratie een belangrijke rol speelt:

- IMX-Geo (Geonovum), Bestuurlijke Gebieden (BZK), Gebouwdossier (Kadaster)
- Diverse Digital Twin use cases (navraag doen bij Bart en/of Koos)
- Digilab use cases: opkopersbescherming (RVIG/VNG), maximale huurverhoging (Belastingdienst)

§ 6.1.4 Aandachtsgebieden van Orkestratie

Interessante onderwerpen die sterk gerelateerd zijn aan orkestratie zij:

- Fouttolerantie (graceful degradation, resiliency, retry policies, automatisch terugmelden).
- Doodlopende links (afwijkingen van informatiemodel, actualiteits-issues, etc.)
- Mapping en herleidbaarheid (zie IMX)
- Historie en tijdreizen (separate module in concept)
- Batching (separate module in concept)
- Orkestratie in OAuth (o.a. Token Exchange, eHerkenning, FSC, etc.)

§ 6.1.5 IMX

Kadaster heeft in samenwerking met Geonovum het IMX initiatief gestart. Dit initiatief heeft als ambitie om basisregistraties (en andere bronnen) in samenhang te kunnen bevragen door middel van API orkestratie. Om op een efficiënte manier te kunnen orkestreren moeten bron API’s voldoen aan diverse randvoorwaarden. De uitdaging is om te onderzoeken welke randvoorwaarden dit zijn en op welke manier deze gestandaardiseerd zouden kunnen worden als design rules.

§ 6.1.6 Arazzo

De Arazzo-specificatie is een nieuwe, door de gemeenschap gedreven standaard die is ontwikkeld onder het OpenAPI-initiatief, met als doel de documentatie en interactie van API's te verbeteren. Het biedt een **programmeertaal-onafhankelijk** kader om reeksen API-aanroepen en hun afhankelijkheden te definiëren, waardoor de communicatie van workflows duidelijker wordt.

Dit project bevindt zich echter nog in een vroeg stadium. Ook is het maar de vraag in hoeverre (commerciële) software vendors erbij gebaat zouden zijn een dergelijke standaard te implenteren. Verder wordt de kanttekening geplaatst dat benodigde stappen bij orkestratie ook dynamisch kunnen worden berekend, zoals bij IMX wordt gedaan. In dat geval is het specificeren van een workflow niet relevant.

6.1.6.1 Belangrijkste Kenmerken

- **Deterministische Workflows**: Arazzo maakt zowel menselijke leesbare als machine-leesbare documentatie mogelijk, wat de ervaring voor API-aanbieders en -gebruikers verbetert.
- **Toepassingen**: Interactieve workflowdocumentatie Automatische documentgeneratie Code- en SDK-generatie op basis van functionele gebruiksgevallen Automatisering van testgevallen en nalevingscontroles AI-gestuurde deterministische API-aanroep

6.1.6.2 Relatie met OpenAPI

Arazzo aanvult de bestaande OpenAPI-specificatie door beschrijvingen van **groepen API's** en hun interacties mogelijk te maken, in plaats van alleen individuele API's. Deze bredere scope ondersteunt automatisering en codegeneratie, met als uiteindelijke doel de waarde die uit API's wordt gehaald te maximaliseren.Het project staat open voor deelname van de gemeenschap, met middelen beschikbaar op GitHub voor degenen die geïnteresseerd zijn in bijdragen of meer willen leren over de specificatie.

6.2 Orkestratie beveiliging

6.2.1 Context

Orkestratie services bieden over het algemeen een oplossing voor een complexe vraag en een antwoord wat input uit meerdere databronnen bevat. Conform de Api Strategie architectuur typologie is het daarmee een zogenaamde "Composite API" die meerdere systeem API's aanroept.

Voor de beveiliging van een dergelijke composite API is het belangrijkste verschil of de API kennis heeft van de inhoud van de gegevensuitwisseling en van die inhoud ook logging bijhoudt of dat de Composite API geen kennis heeft van de inhoud en daarmee ook geen logging hoeft bij te houden.

Wanneer de Composite API geen kennis heeft van de inhoud en geen logging vasthoud noemen we dit transparant.

> we focussen in deze context op het bevragen van services en niet op de transactionele kant
>
> We gaan in onderstaande situaties er vanuit dat er vertrouwelijke gegevens worden bevraagd. Voor het bevragen open data is dergelijke beveiliging niet noodzakelijk.
>
> We gaan er van uit dat OAuth wordt gebruikt voor de authorisatie van de Services.

6.2.2 Use cases

In deze context onderkennen we vier use cases

- De Client bevraagt direct de bronnen en heeft de orkestratie daarmee ingebed in de client software (deze use case wordt verder niet uitgediept)
- De User Identity wordt onderbroken door de orkestratie (Implicit Identity reuse)
- De User Identity propageert door tot de bronnen (Identity propagation)
- De user bevraagt 1 bron en de service van de bron bevraagt nog een andere bron zonder dat de user of client hier weet van heeft (deze use case wordt verder niet uitgediept)

6.2.3 Begrippen

- OAuth:
- Identity propagation:
- Identity reuse:
- Vertrouwelijke gegevens:
- Open Data:
- Token Exchange:
- Identity Service Provider:

6.2.4 Implicit Identity reuse - token based

6.2.4.1 Rationale

- er is een ontkoppeling tussen de Services en de client. Dit is geen probleem bij open data maar wel bij vertrouwelijke gegevens.
- In de praktijk is dit wel hoe bijvoorbeeld een huisarts werkt en hoe veel balies en overheidsorganisaties werken.
- De token / OAuth flow die wordt geschetst kan in deze situaties ook een andere vorm van authenticatie of identificatie zijn
- Deze situatie is vaak in gebruik binnen 1 organisatie waarbij alle onderdelen van de orkestratie onder verantwoording van 1 organisatie vallen en de User ook in dienst is bij deze organisatie

6.2.4.2 Identity

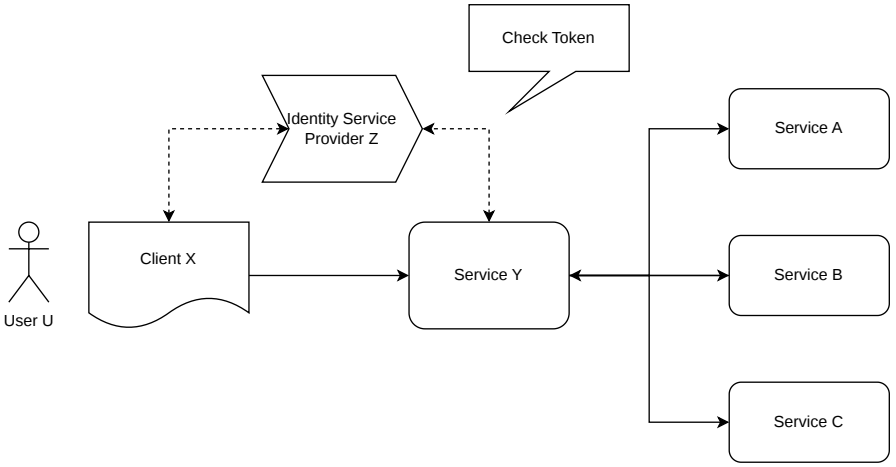
- Service A,B & C kennen de identity van Service Y, niet van Client X of User U
- Service Y kent de Identity van User U en of Client X

6.2.4.3 Authenticiteit

- De Authenticiteit van Client X wordt gegarandeerd door Identity Service Provider Z
- De Authenticiteit van Client X wordt geverifieerd door Service Y en Identity Service Provider Z
- De Authenticiteit van Service Y moet worden geverifieerd door Client X
- De Authenticiteit van Service A,B & C moet worden geverifieerd door Service Y

6.2.4.4 Autorisatie

- De autorisatie is bepaald in en door de Service provider Y, Service A,B & C weten niet dat hun gegevens wordt gedeeld met Client X en User U of een andere client of user
- Service Y moet geautoriseerd zijn bij Service A,B & C om namens Client X en/of User U gegevens op te vragen
- Service A,B & C delegeren toegang tot hun gegevens aan Service Y



§ *6.2.4.5 Implicaties*

- Er is een sterke vertrouwensband nodig tussen zowel de Client & Service alsook de Service en de Services. Deze vertrouwensband zal waarschijnlijk worden vertaald naar eisen, audits en contracten.
- De aanbieder van Service Y, stel een gemeente, zal zorgvuldig moeten borgen en vastleggen dat de service die ze aanbieden alleen gegevens opvraagt bij de Services wanneer daar ook een expliciete vraag voor is van de User U. Er moet worden voorkomen dat in deze situatie een kwaadwillende toegang krijgt tot de service en daarmee alle Services van alle users kan bevragen.
- Deze oplossing is technisch waarschijnlijk makkelijker te realiseren en vaker in gebruik maar kent ook veel grotere risico's dan de transparante orkestratie.

§ **6.2.5 Identity propagation - token based**

§ *6.2.5.1 Rationale*

Er is een direct verband tussen de Services en de identity van de client/gebruiker. Dit is maakt het mogelijk om ook privacy vertrouwelijke gegevens veilig te gebruiken in de orkestratie.

- In de praktijk wordt deze vorm van orkestratie ook vaak client side gedaan door vanuit de client direct meerdere API's aan te roepen,
- en is er bij de orkestratie service vaak geen sprake van filtering van gegevens,
- het betreft dan meer een bundeling van meerdere bronnen dan echt orkestratie,
- het lijkt bij bevragingen vaak meer op het composition patroon dan een orkestratie patroon.

§ *6.2.5.2 Identity*

- Service A,B&C kennen de Identity van user U (de recources zijn namelijk aan de User gerelateerd of de User is vanuit zijn functie gemachtigd de Services te raadplegen)
- Service Y kent de Identity van user U niet

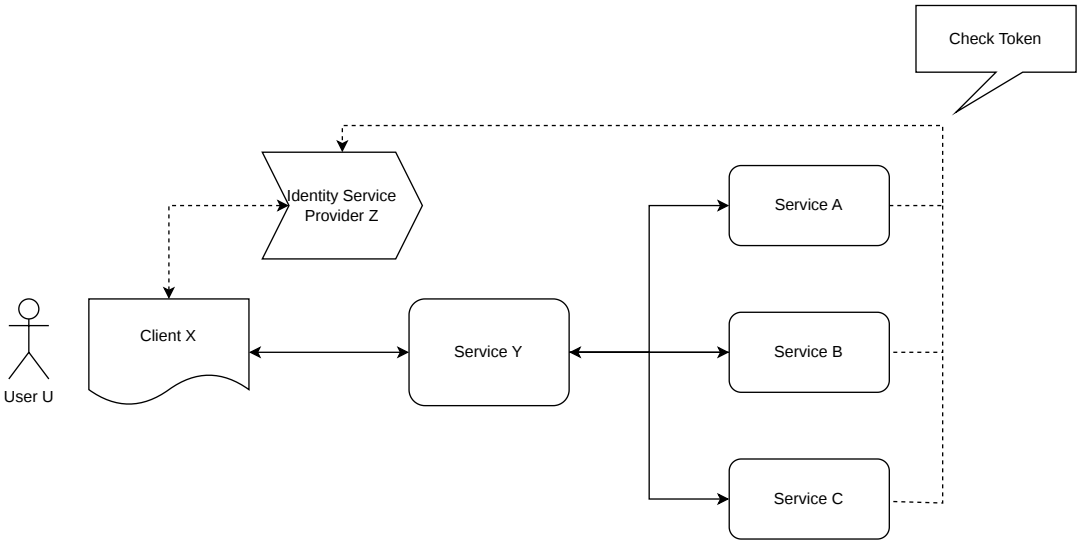
§ *6.2.5.3 Authenticiteit*

- De Authenticiteit van Client X wordt gegarandeerd door Identity Service Provider Z
- De Authenticiteit van Service Y wordt geverifieerd door A,B & C
- De Authenticiteit van Service Y moet worden geverifieerd door Client X

- De Authenticiteit van Service A,B & C moet worden geverifieerd door Service Y

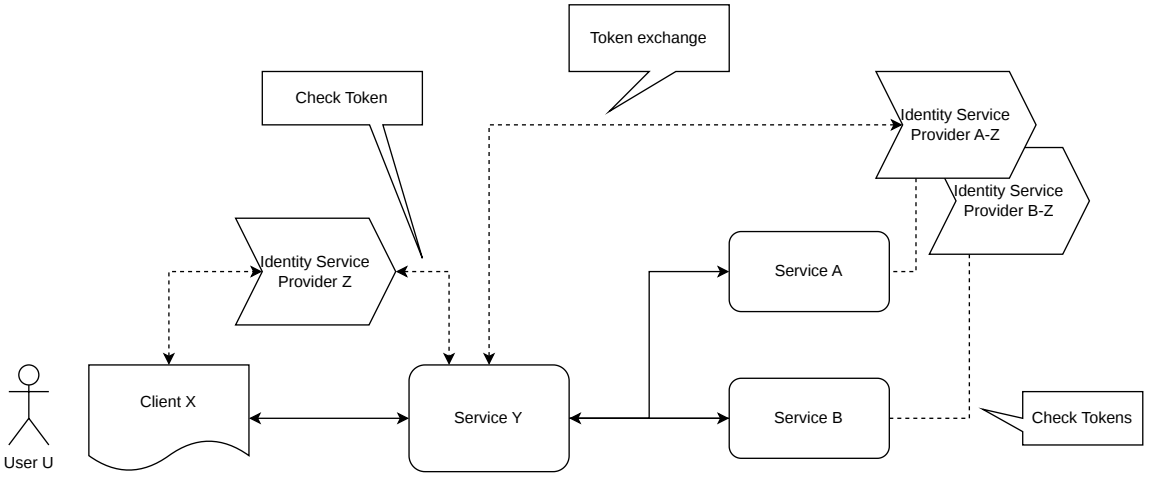
§ *6.2.5.4 Autorisatie*

- De autorisatie is per Service dynamisch in te stellen
- Service Y moet overweg kunnen met het feit dat Service A,B & C de autorisatie kunnen weigeren



§ **6.2.6 Variatie - Federated Identity Propagation**

1. Identity Service Provider Z kan ook de toegang zijn tot een gefedereerde omgeving van Identity providers waarbij Z door middel van OAuth Token Exchange de Access tokens worden opgehaald bij de Identity providers van de verschillende Services en deze tokens door de client worden meegegeven bij het request



§ *6.2.6.1 Implicaties Identity propagation*

- Er is een sterke vertrouwensband nodig tussen zowel de Client & Service alsook de composite Service en de system Services. Deze vertrouwensband zal waarschijnlijk worden vertaald naar eisen, audits en contracten om te voorkomen dat de Service Y toch de gegevens inziet die de resouces terugsturen aan de client.
- Organisatorisch zullen X, Y, Z, A & B praktisch gezien niet allemaal van verschillende organisaties zijn. Het is logischer wanneer A, B & Y van 1 organisatie zijn of wanneer X & Y van 1 organisatie zijn. Service Y wordt namelijk alleen ontwikkeld als er een vraag is en een voordeel voor de Services of de client.

- Afhankelijk van de situatie is het waarschijnlijk dat Identity provider Z een algemene dienst is van de overheid (denk aan DigiD of eHerkenning) of dat dit een dienst is van 1 van de organisaties in de orkestratie.

6.2.7 Conclusies en Adviezen

- Voor een moderne API architectuur is een identity service provider cruciaal.
- Niet alleen om IAM toe te passen op de eigen Service maar ook om achterliggende services aan te kunnen spreken.
- Met de groei van het aantal system APIs en vraag naar Composite APIs groeit het beveiligingsvraagstuk exponentieel mee.
- Bereid API's voor op de IAM mbv Oauth.
- Houd er rekening mee en faciliteer token exchange als onderdeel van je roadmap / groei / maturity level.

A. Referenties

A.1 Informatieve referenties

[rfc8693]
OAuth 2.0 Token Exchange. M. Jones; A. Nadalin; B. Campbell, Ed.; J. Bradley; C. Mortimore. IETF. January 2020. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8693

↥