AzureDay

Poland

OAuth 2.0 Exemplified

Marek Grabarz | Technology Advisor

March 3rd

2020

AzureDay

Poland

Don't forget to rate my session in Mobile App!

AzureDay
Poland

Mobile App

March 3rd
2020

AzureDay
Poland

March 3rd
2020

Microsoft Azure User Group Poland

Microsoft

accenture

allegro Tech

BLUESOFT EFFICIENT SOLUTIONS

Chmurowisko

Demant

dynatrace

psc_ TRANSITION

TechData

T · · Systems ·

Session agenda

- Introduction to OAuth 2.0
- More about Outh 2.0 in Azure
- Which grant do I need?
- Grants in detail
  - Client credentials
  - Authorization code (+PKCE)
  - Resource owner password credentials
  - Implicit
  - Delegation scenarios

Introduction to OAuth 2.0

Many luxury cars today come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will not allow the car to drive more than a mile or two. Some valet keys will not open the trunk, while others will block access to your onboard cell phone address book. Regardless of what restrictions the valet key imposes, the idea is very clever.

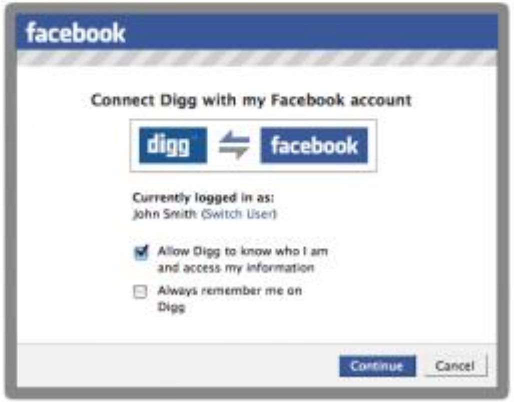
You give someone limited access to your car with a special key, while using your regular key to unlock everything.

OAuth's website

OAuth - stateless authorization protocol

- Allows you to authorize service to acess your resources in another service
- Access is granted to token bearers.
- Decouples tokens from Clients (when compared to OAuth 1.0)

OAuth is everywhere and is cloud born



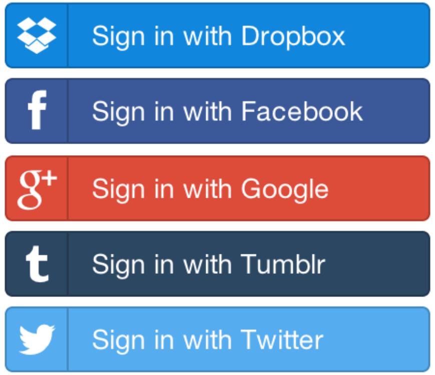
facebook

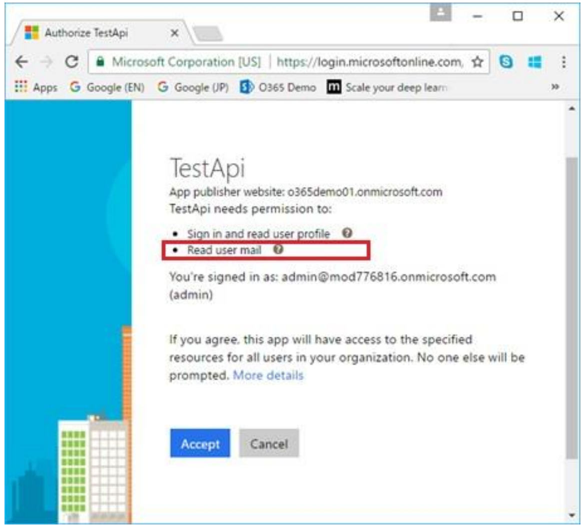
Connect Digg with my Facebook account

Currently logged in as:
John Smith (Switch User)

Allow Digg to know who I am and access my information

Always remember me on Digg





Authorize TestApi

Microsoft Corporation [US] | https://login.microsoftonline.com

TestApi

App publisher website: o365demo01.onmicrosoft.com

TestApi needs permission to:

- Sign in and read user profile
- Read user mail

You're signed in as: admin@mod776816.onmicrosoft.com (admin)

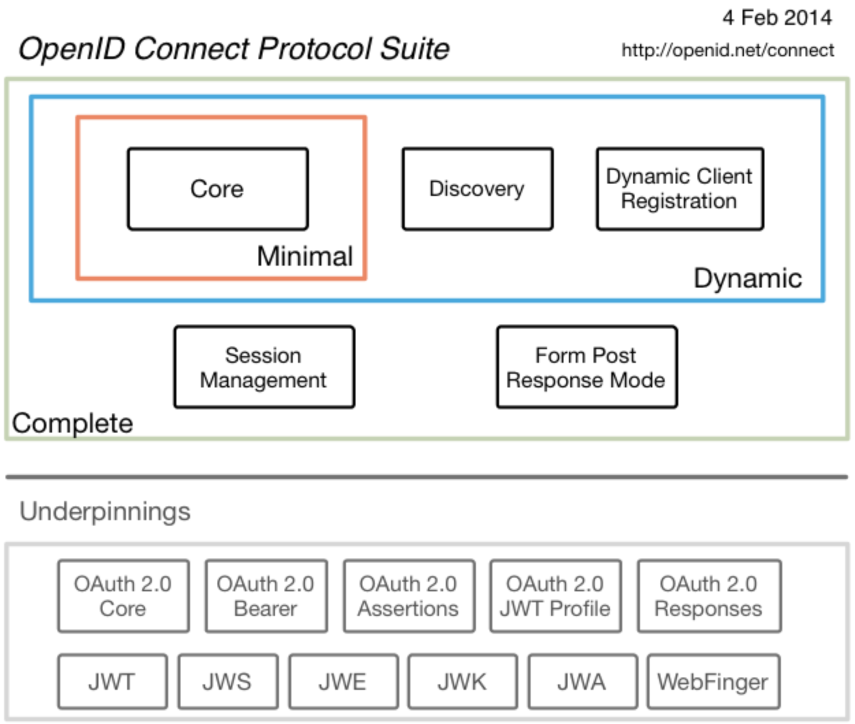
If you agree, this app will have access to the specified resources for all users in your organization. No one else will be prompted. More details

OAuth 2.0 roles

- Resource Owner (the User)
- Resource Server (the API)
- Client (the App - native/website)
- Authorization Server (the Server granting access)

OpenID Connect (OIDC)

- Authentication protocol
- Identity layer on top of OAuth 2.0
- Gives one login to many apps
- Provides user claims for tokens



Token types

- Access Token
  - Used to make authenticated calls to Resource Server
  - Represents that Resource Owner authorized Client to access Resource Server
  - Contains time information, scopes, audience

- Refresh Token
  - Used to get new Access Token when previous one expires
  - ID Token
  - Contains information about the Resource Owner

Access token format

- JWT (JSON Web Token)
- Three parts – header, payload, signature
- All parts in base64 separated by „.”
- Base64OfHeader.Base64OfPayload.Base64OfSignature
- User with Bearer Authorization schema
- https://jwt.io https://jwt.ms

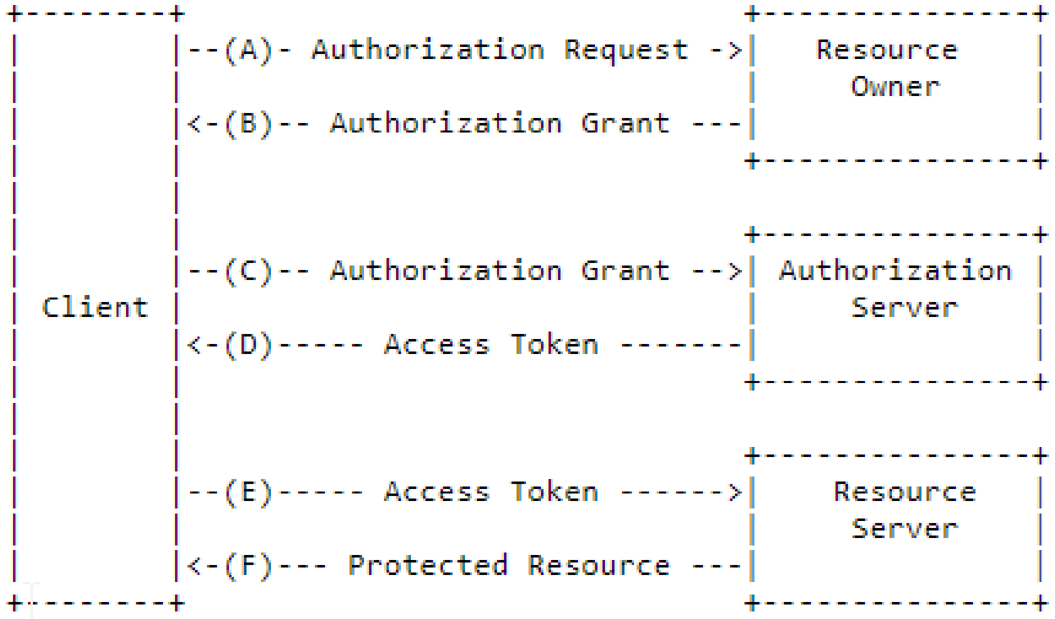
Claims

- The information contained in JWTs are known as "claims", or assertions of information about the bearer and subject of the token.
- OIDC claims http://openid.net/specs/openid-connect-core-1_0.html#Claims
- Access Token Claims: aud, iss, sub, iat, nbf, exp, nonce ☺

Token endpoints (Authorization Server)

- Authorization Enpoint – where Resource Owner allows Client to access Resource Server
- Redirect Enpoint – where Authorization Entopoint returns back
- Token Enpoint – where Client obtains Access Token using authorization given from Resource Owner
- More… UserInfo, Logout, Discovery, Revocation…

Simplified flow



```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

More about OAuth 2.0 in Azure

Azure AD, Azure AD v2, Azure AD B2C

OAuth in Azure AD

- AAD currently support two distinct OAuth enpoints
- /oauth2/v2.0/token and /oauth/token
- Two separate portals for application management and registration
- ADAL vs MSAL

- well_known: https://login.microsoftonline.com/TenantID/v2.0/.well-known/openid-configuration

Applications in AAD

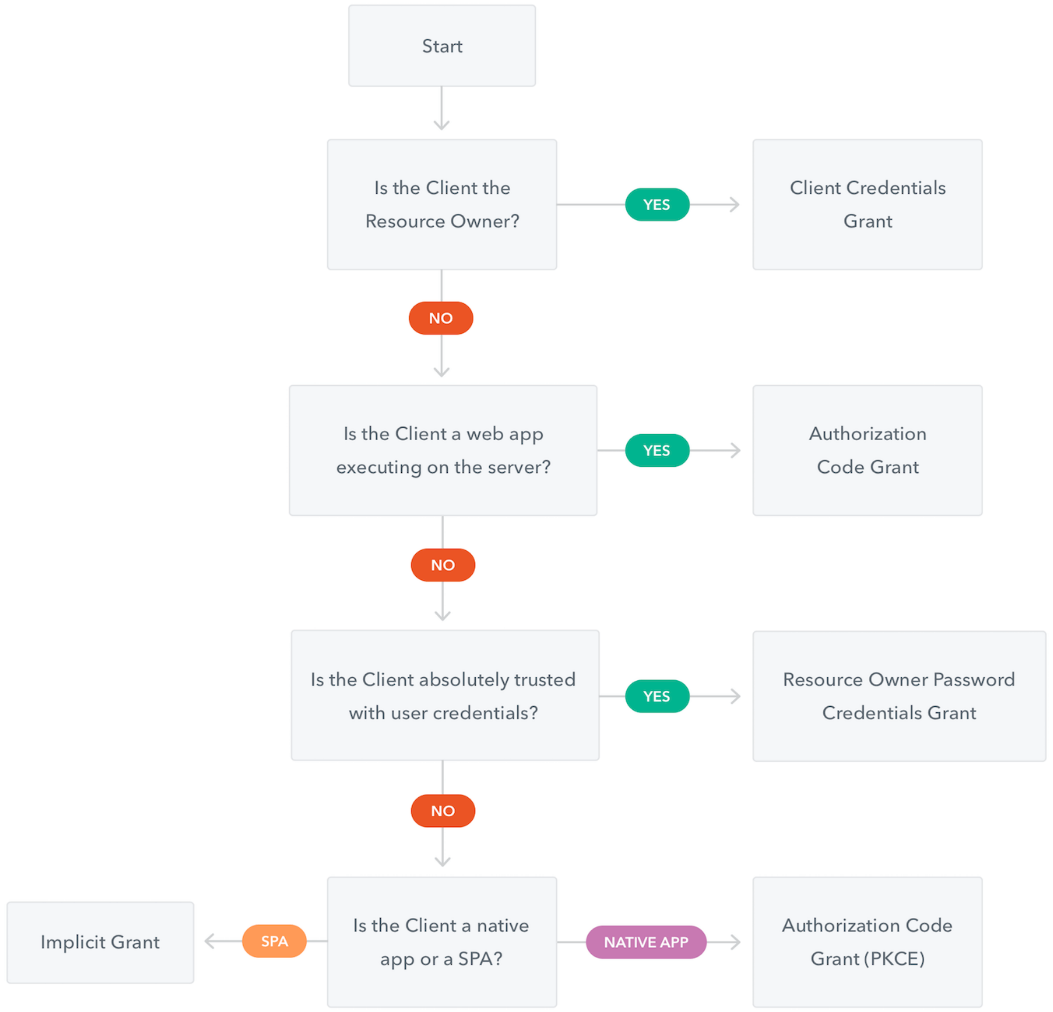
- App types (native and Web/API)
- Registration portals
- Enterprise Applications
- Permissions
- Consents

OAuth in AAD B2C

- B2C only supports v2 endpoints
- /oauth2/v2.0/token
- Dedicated B2C blades, but there is still need fo AAD blade in limited cases.
- MSAL only

Which flow do I need?

All of them (almost)


Start
Is the Client the Resource Owner?
YES
Client Credentials Grant
NO
Is the Client a web app executing on the server?
YES
Authorization Code Grant
NO
Is the Client absolutely trusted with user credentials?
YES
Resource Owner Password Credentials Grant
NO
Is the Client a native app or a SPA?
SPA
Implicit Grant
NATIVE APP
Authorization Code Grant (PKCE)

Client Credentials Grant

- Used when Client is accessing:
  - Protected resources under its control
  - Or resources owned by others, that have been previously arranged to get access to.

- You will find this flow useful for:
  - Console apps, daemons, non-interactive tools
  - Apps that authenticate via service principal (client id and secret). No password used.

Authorization Code grant and JWT Bearer grant

- Used when Client is accessing:
  - Resources owned by other Resource Owner via impersonation.
  - Credentials of Resource Owner are not disclosed to Client

- You will find this flow useful for:
  - Webistes/APIs calling other APIs in context of logged in user.
  - Requires interactive logon or SSO.

Resource Owner Password Credentials Grant

- Used when Client is accessing:
  - Resources owned by other Resource Owner via impersonation.
  - Credentials of Resource Owner are stored by Client or processed by Client
- You will find this flow useful for:
  - Console apps, daemons, non-interactive tools – even tests
  - Apps that authenticate as User, with his login and password
  - Apps need to be highly trusted

Implicit Grant and ACG PKCE

- Used when Client is accessing:
  - Resources owned by other Resource Owner via impersonation.
  - Client app is public, with no secret.

- You will find this flow useful for:
  - SPA webistes
  - Desktop, mobile apps distributed anywhere
  - Consider PKCE instead of Implicit.

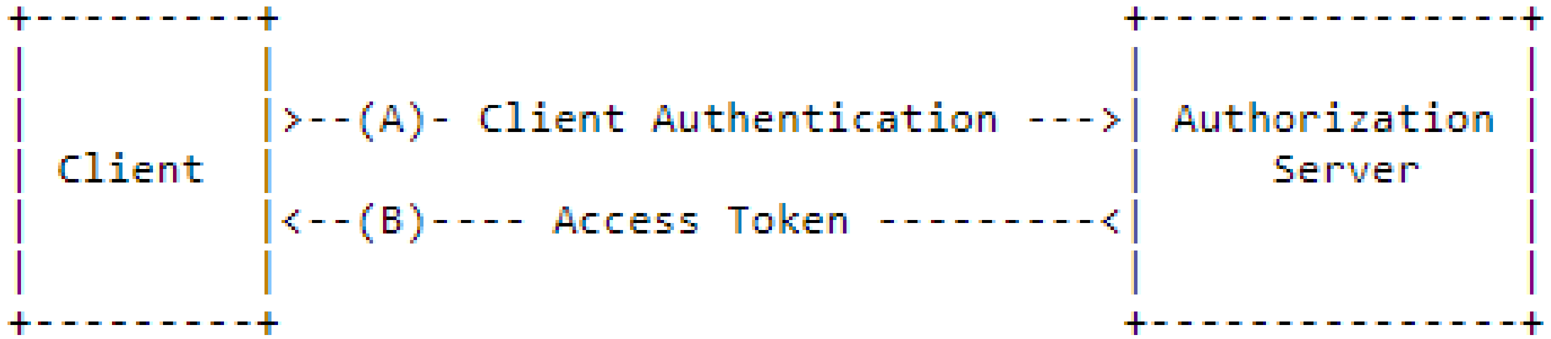
Is my flow supported?

	Client Credentials	Authorization Code	Resource Owner Password Credentials	Implicit	On Behalf Of
Azure AD	👍	👍	👍	👍	👍
Azure AD v2	👍	👍		👍	👍
Azure AD B2C	👍?	👍	👍	👍	

Flows in detail…

Keep your seatbelts fastened

Client Credentials Grant - Diagram



```
+---------+                                  +---------------+
|         |                                  |               |
|         |>--(A)- Client Authentication --->| Authorization |
| Client  |                                  |     Server    |
|         |<--(B)---- Access Token ---------<|               |
|         |                                  |               |
+---------+                                  +---------------+
```

Client Credentails Grant - Ingredients

- Client is confidential
- One POST request to /Token endoint
- Need to add client_id, client_secret, grant_type=client_credentials and scope.
- No refresh_token
- No delegated permissions

Authorization Code Grant - Diagram

```
+----------+
| Resource |
|   Owner  |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier      +---------------+
|         -+----(A)-- & Redirection URI ---->|               |
|  User-   |                                 | Authorization |
|  Agent  -+----(B)-- User authenticates --->|     Server    |
|          |                                 |               |
|         -+----(C)-- Authorization Code ---<|               |
+-|----|---+                                 +---------------+
  |    |                                         ^      v
 (A)  (C)                                        |      |
  |    |                                         |      |
  ^    v                                         |      |
+---------+                                      |      |
|         |>---(D)-- Authorization Code ---------'      |
|  Client |          & Redirection URI                  |
|         |                                             |
|         |<---(E)----- Access Token -------------------'
+---------+       (w/ Optional Refresh Token)
```

Authorization Code Grant - Ingredients

- Both public (with PKCE) and confidential clients
- GET request to /authorize and receive authorization_code. Another POST request to /token
- Need to send client_id (client_secret), redirect_uri, grant_type=authorization_code
- Supports all token types
- Both delegated and client permissions

Authorization Code Grant - PKCE

- Defined in RFC7636
- Mainly addresses a risk of intercepting authorization code in mobile apps (but not only… https://tools.ietf.org/html/draft-ietf-oauth-security-topics-11)
- PLAIN: code_challenge = code_verifier
- S256: code_challenge = BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))

Resource Owner Password Credentials Grant - Diagram

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     v
     |    Resource Owner
    (A) Password Credentials
     |
     v
+---------+                                  +---------------+
|         |>--(B)---- Resource Owner ------->|               |
|         |         Password Credentials     | Authorization |
| Client  |                                  |     Server    |
|         |<--(C)---- Access Token ---------<|               |
|         |    (w/ Optional Refresh Token)   |               |
+---------+                                  +---------------+
```

Resource Owner Password Credentials Grant - Ingredients

- Public client only!
- Single POST request to /token endpoint.
- Need to send grant_type=password, login, password, client_id, scope.
- No refresh_token

Implicit Grant - Diagram

```
 +----------+
 | Resource |
 |  Owner   |
 |          |
 +----------+
      ^
      |
     (B)
 +----|-----+          Client Identifier     +---------------+
 |         -+----(A)-- & Redirection URI --->|               |
 |  User-   |                                | Authorization |
 |  Agent  -|----(B)-- User authenticates -->|     Server    |
 |          |                                |               |
 |          |<---(C)--- Redirection URI ----<|               |
 |          |          with Access Token     +---------------+
 |          |            in Fragment
 |          |                                +---------------+
 |          |----(D)--- Redirection URI ---->|   Web-Hosted  |
 |          |          without Fragment      |     Client    |
 |          |                                |    Resource   |
 |     (F)  |<---(E)------- Script ---------<|               |
 |          |                                +---------------+
 +-|--------+
   |    |
  (A)  (G) Access Token
   |    |
   ^    v
 +---------+
 |         |
 |  Client |
 |         |
 +---------+
```

Implicit Grant - Ingredients

- Do not use it ;-)
- Public client only
- GET request to /Authorize endpoint
- Need to add client_id, response_type=token, scope, redirect_uri
- No refresh_token
- Only delegated non-admin permissions

Delegation scenarios

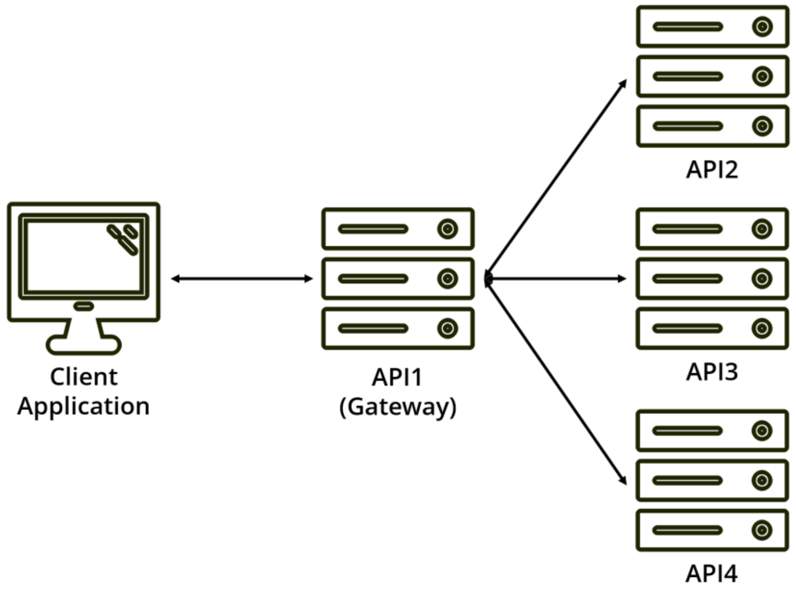
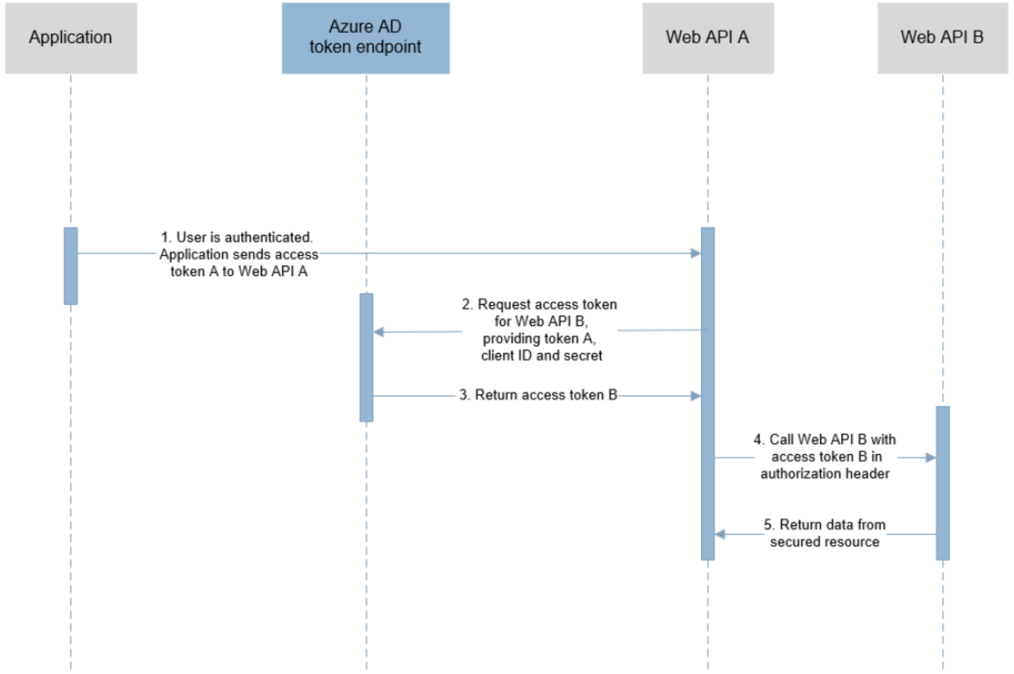


Image: by Scott Brady
(https://scottbrady91.com)

Delegation scenarios

- Passing token
- Client credential grant in the middle.
- JWT Bearer Authorization Grant (RFC 7523), same for SAML (RFC 7522)
- Custom grants – Azure's JWT Bearer grant!!!
- OAuth token exchange!!! https://tools.ietf.org/html/rfc8693

JWT Bearer Grant - Diagrams


Application
Azure AD token endpoint
Web API A
Web API B
1. User is authenticated. Application sends access token A to Web API A
2. Request access token for Web API B, providing token A, client ID and secret
3. Return access token B
4. Call Web API B with access token B in authorization header
5. Return data from secured resource

JWT Bearer Grant - Ingredients

- For confidential application.
- Similar to authorization code grant, but with second /token POST request.
- Need to send grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer, client_id, client_secret, requested_token_use=on_behalf_of, assertion={access_token}
- Same tokens/permissions as ACG

AzureDay

Poland

Don't forget to rate my session in Mobile App!

AzureDay

Poland

Mobile App

March 3rd

2020