Paweł Rajba
pawel@ii.uni.wroc.pl
http://kursy24.eu/

Application Security
OpenID Connect

Agenda

- Authentication & OAuth2
- OpenID Connect
  - Request
  - Response
  - Rules
  - Profiles
  - Discovery and dynamic registration
- Playground

Authentication & OAuth2

- It is quite popular that OAuth2 is abused for authentication
- The most common scenario is as follows:
  - User authenticates on AS
  - Afterwards an application exchange code for access token
  - The assumption is that if the application is able to get data using access token, then it means that user properly authenticated on AS

Authentication & OAuth2

- Where are the problems here?
  - OAuth2 is an authorization framework, there is no flow related to authentication
    - Although authentication is a part of the OAuth2 flow
  - The focus is on the client application, not on a user
    - In other words, authorization is for the client application, not for the user
    - After getting access token, user is no more involved
  - In a standard implementation, there is only authorization information
    - There is no information about the user
  - When someone has a token, has the access
    - There is no additional verification e.g. who is the proper receiver of the token

Authentication & OAuth2

- The main problem:
  - Access token gives an application access to the scope related to the token
    - What means whoever has the token is able legally perform operation
  - After authentication we expect, that request is performed only by authenticated user itself
    - It is not possible in the OAuth2, because if another application take over the token, it can still access services

Authentication & OAuth2

- Very good consideration
  - http://www.cloudidentity.com/blog/2013/01/02/oauth-2-0-and-sign-in-4/
  - http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html

Source: http://nat.sakimura.org/2012/01/20/openid-connect-nutshell/

OpenID Connect

- The solution is the OpenID Connect
  - An authentication protocol built on top of OAuth2
    - We can consider OpenID Connect as a OAuth2 profile which defines a flow for authentication
  - Allows to get the information about the user
    - Adds ID Token where this information is stored
  - Emerging protocol, but has many implementations
    - Google is probably the best one
  - The main website: http://openid.net/connect/
  - A very good introduction
    - http://nat.sakimura.org/2012/01/20/openid-connect-nutshell/
- Let's the presentation video
  - https://www.youtube.com/watch?v=Kb56GzQ2pSk
    - We will use the offline mode ☺

OpenID Connect Request

- To make a request the following information is required
  - Client ID
  - Client Secret
  - End-user authorization endpoint
  - Token endpoint
  - User info endpoint
- Additionally:
  - grant_type = token id_token
  - scope = openid profile email …

OpenID Connect Request

- GET
  - /authorize?grant_type=token%20id_token&scope=openid%20proflie&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
    HTTP/1.1
  - Host: server.example.com

OpenID Connect Response

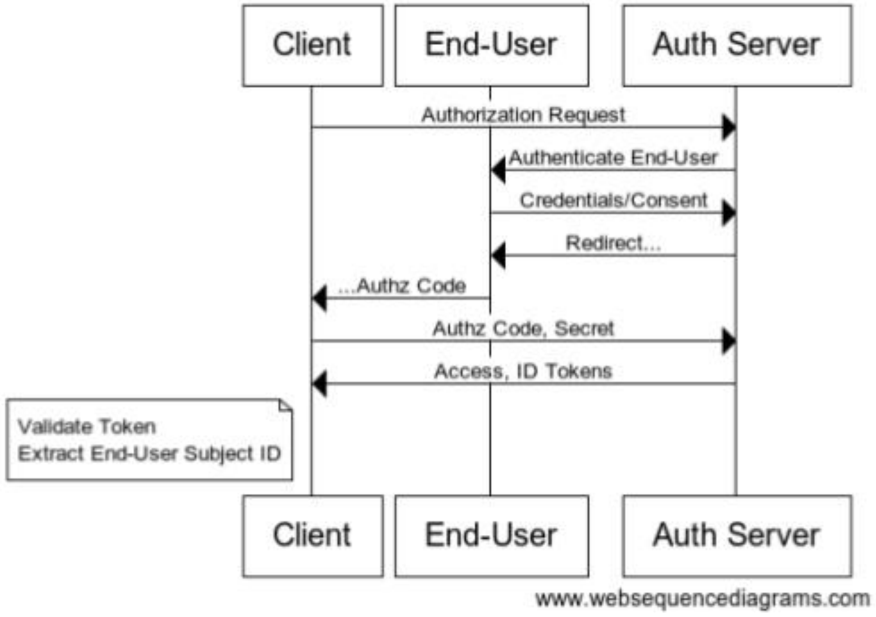
- Beside access_token included in OAuth2 response, one gets id_token with the following information
  - aud (audience)
    - The client_id that this id_token is intended for.
  - exp (expiration)
    - The time after which this token must not be accepted
  - sub (subject)
    - A locally unique and never reassigned identifier for the user (subject)
    - E.g. “24400320” or “AItOawmwtWwcTok51BayewNvutrJUqsvl6qs7A4”.
  - iss (issuer)
    - A https: URI specifying the fully qualified host name of the issuer, which when paired with the user_id, creates a globally unique and never reassigned identifier.
    - E.g. “https://aol.com”, “https://google.com”, or “https://sakimura.org”.
  - nonce - nonce value sent in the request.
- All these parameters are required

OpenID Connect Rules

- The following rules should be applied
  - An authorization server must only issue assertions about user identifiers within its domain
  - The client MUST verify that the aud matches its client_id and iss matches the domain (including sub-domain) of the issuer of the client_id
  - The authorization server is responsible for managing its own local namespace and enforcing that each user_id is locally unique and never reassigned
  - When the client stores the user identifier, it MUST store the tuple of the user_id andiss.
    The user_id MUST NOT be over 255 ASCII characters in length

OpenID Connect Profiles

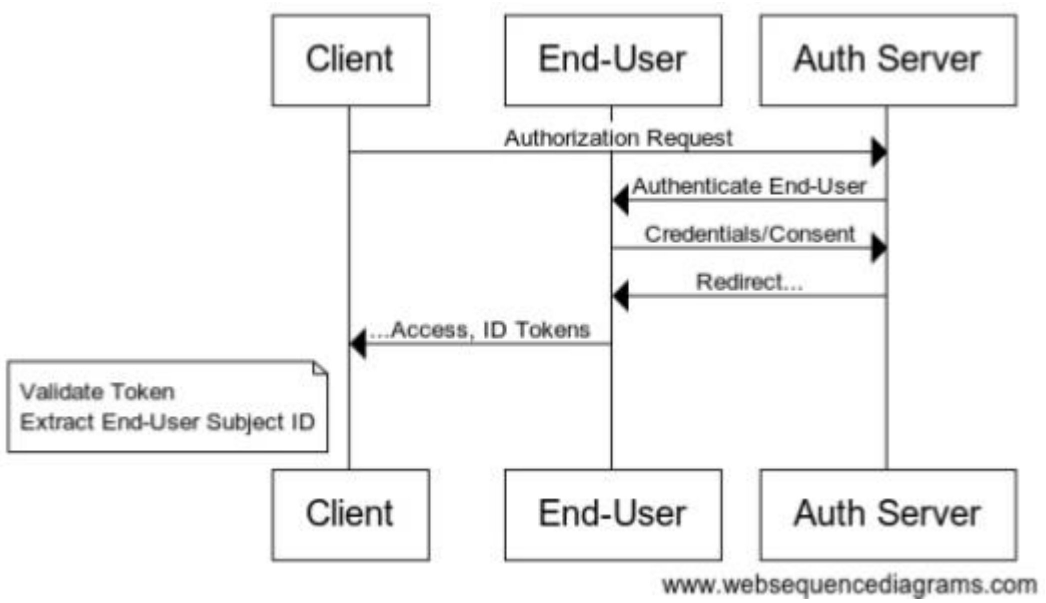
- Basic Client Profile
  - Based on OAuth2 code flow
  - Designed for a web-based relying parties
  - Subset of OpenId Connect Core specification
  - More: http://openid.net/specs/openid-connect-basic-1_0.html



Source: http://www.slideshare.net/metadaddy/openid-connect-an-overview

OpenID Connect Profiles

- Implicit Client Profile
  - Based on OAuth2 implicit flow
  - Designed for a web-based relying parties
  - Subset of OpenId Connect Core specification
  - More: http://openid.net/specs/openid-connect-implicit-1_0.html



Source: http://www.slideshare.net/metadaddy/openid-connect-an-overview

OpenID Connect Discovery and dynamic registration

- Discovery
  - Allows client app to
    - determine the identity of the End-User
      - Based on authentication performed in Authorization Server
    - obtain a basic profile a of End-User
  - Uses WebFinger (RFC7033)
  - More: https://openid.net/specs/openid-connect-discovery-1_0.html
- Registration
  - Allows client app to register on the server
  - More: http://openid.net/specs/openid-connect-registration-1_0.html

OpenID Connect Playground

- A very good open source provider and a set of samples
  - http://thinktecture.github.io/
- Getting started videos
  - Provider introduction
    - http://vimeo.com/91397084
  - Walkthrough samples
    - http://vimeo.com/91405115
- DEMO