Paweł Rajba
pawel@cs.uni.wroc.pl
http://itcourses.eu/

Information Systems Security

OAuth2

Agenda

- Security tokens
  - Tokens history
  - JSON Web Token
- Why tokens?
- OAuth2?
  - Actors
  - Client types and profiles
  - Registration
  - Authorization flows

Security tokens

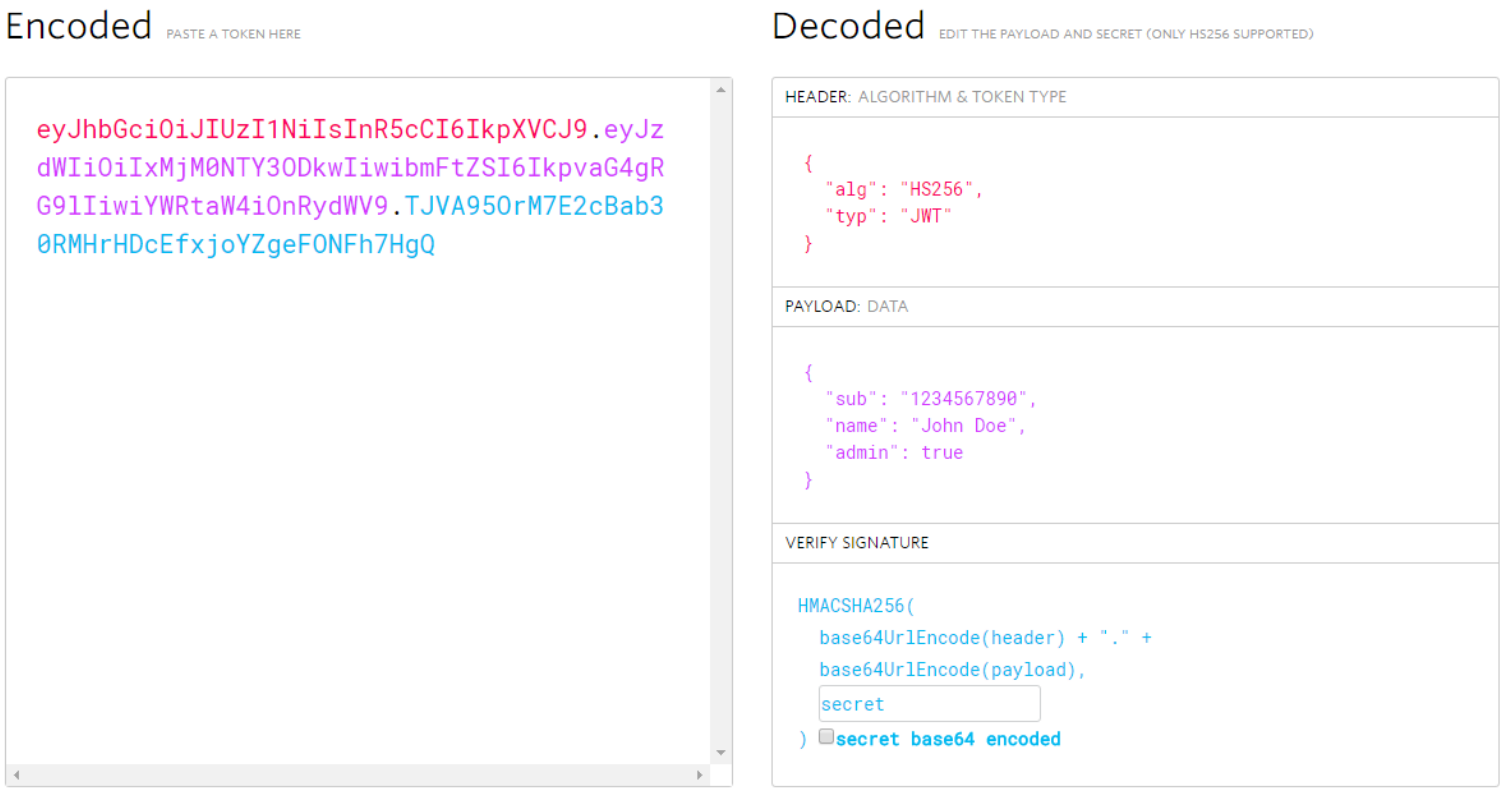
- A data structure with the following features
  - Contains information about an issuer and a subject, usually with expiration date
  - Signed, sometimes also encrypted
  - Typical roles
    - A client requests a token
    - An issuer issues a token
    - A service consumes a token
      - There is a trust between the issuer and the service

Security tokens

- Tokens history
  - SAML 1.1/2.0
    - XML based format
    - Very expressive with many options, including security
    - Popular in SOAP services
  - Simple Web Token (SWT)
    - Form/URL based format
    - Very limited possibilities, e.g. only symmetric signatures
  - JSON Web Token (JWT)
    - JSON based format
    - A new format with a strongly increasing prevalence
    - Lightweight, however quite expressive
      - But still SAML is much more expressive

Security tokens

- JSON Web Token



- Let's take a look on https://jwt.io/

Security tokens

- All parts are encoded with Base64url:
- Base64 vs Base64url
  - Both are intended to encode binary data into ASCII
  - However, Base64url is intended to be URL safe
    - „+" is replaced by „-"
    - „/" is replaced by „_"
    - Padding „=„ is usually ommitted
      - optional, but not recommended
- More: http://en.wikipedia.org/wiki/Base64

Security tokens

- JSON Web Token, claims
  - There are 3 sets of claims
    - Registered in IANA (like iss, iat, exp, …)
    - Public claim name
    - Private claim name
  - Common claims
    - "iss" (Issuer)
    - "sub" (Subject)
    - "aud" (Audience)
    - "exp" (Expiration Time)
    - "nbf" (Not Before)
    - "iat" (Issued At)
    - "jti" (JWT ID) Claim
- Documentation
  - http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html

Security tokens

- More about standards
  - JSON Web Algorithms (JWA)
    - Details on algorithms around the JWT, JWS, JWE, JWK
  - JSON Web Key (JWK)
    - Data structure represting keys for singing and encryption
  - JSON Web Token (JWT)
    - Data structure for representing claims
  - JSON Web Encryption (JWE)
    - Encrypted JWT
  - JSON Web Signature (JWS)
    - Signed JWT
- Corollary: a JWT on slide 5 was actually JWS

Why tokens?

- We consider 2 main approaches for granting access:
  - Cookie-based authentication
    - In a cookie is only session ID
    - Whole information about an user is in session on a server
  - Token-based authencation
    - Whole information about an user is in token
    - There is no session needed – authN is stateless

Why tokens?

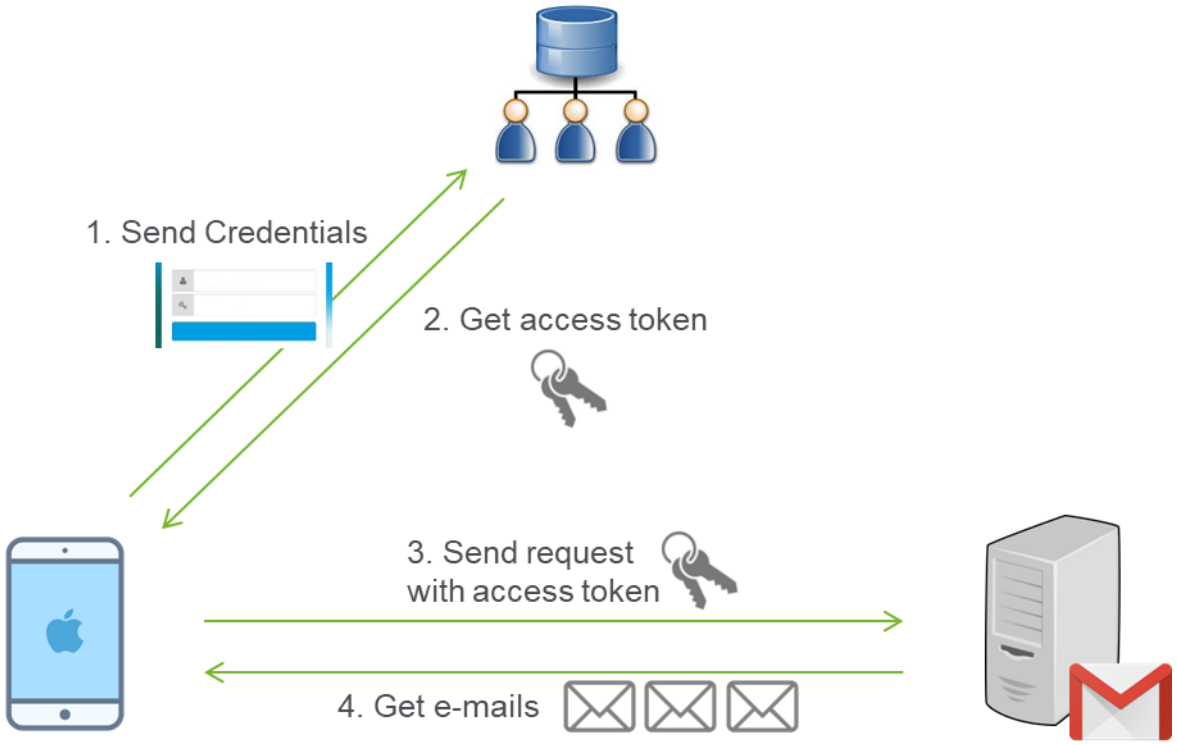
- What main arguments do we have for tokens?
  - Cross-domain
    - If we use HTTP header, cross domain is easily achievable
  - Stateless
    - No session is needed
  - Single Responsilibity
    - Granting access process is separated from serving data
    - There is no coupling between token issuer and consumer
  - Mobile compatible
    - Most of current mobile technologies are tokens-oriented

OAuth2

- Let's imagine the following scenario
  - You have an account on Google
  - You found a very fancy calendar application on your phone market
  - You want to use it, but don't want to give the application permission to all Google account data (e.g. mails, contacts, etc. – only calendar entries)
- In this scenario we consider 3rd party application which is considered as untrusted
  - And this is the place when the OAuth2 helps

OAuth2

- There is a nice iOS application to show e-mails: **GimmeMails**
  In OAuth2 the flow looks a follows:



OAuth2

In the framework there is a service where you can authenticate, but in the return application gets an *access token* which allows the application to access specific data

There is no user involved after authentication

OAuth2

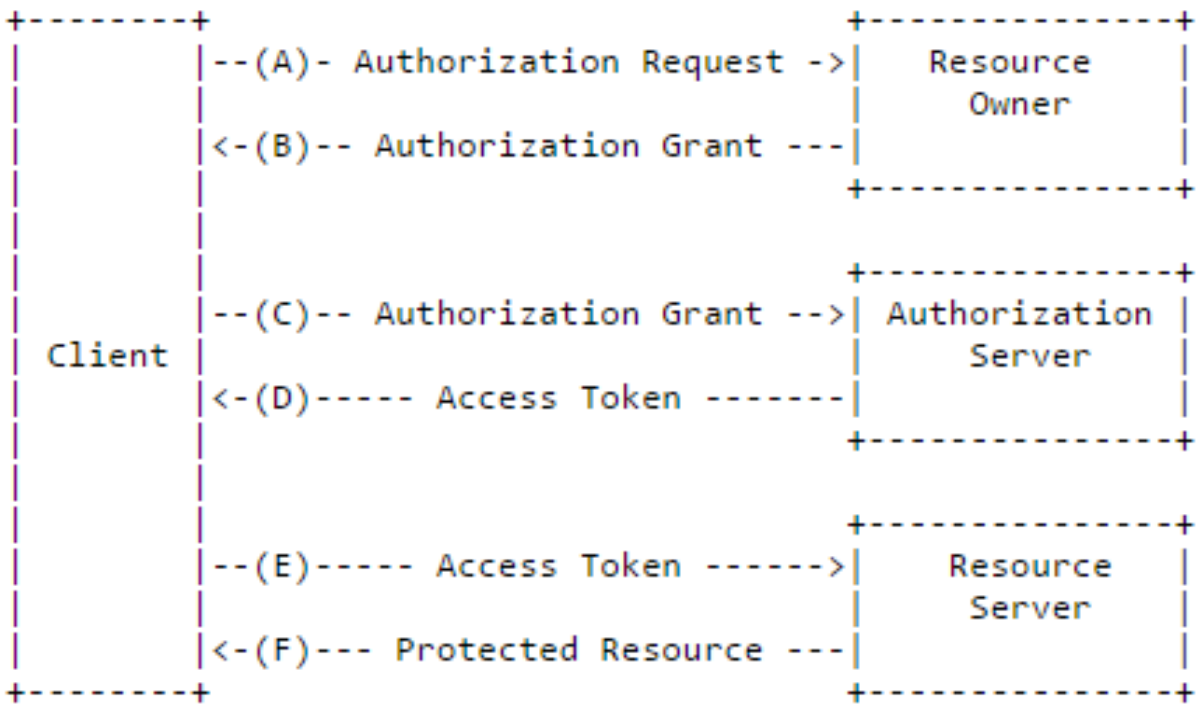
- Described in RFCs:
  - The OAuth 2.0 Authorization Framework
    - https://tools.ietf.org/html/rfc6749
  - The OAuth 2.0 Authorization Framework: Bearer Token Usage
    - https://tools.ietf.org/html/rfc6750
  - OAuth 2.0 Dynamic Client Registration Management Protocol
    - https://tools.ietf.org/html/rfc7592
  - OAuth 2.0 Token Introspection
    - https://tools.ietf.org/html/rfc7662

OAuth2

- Actors
  - Resource server
    - Service which is protected and understands tokens
  - Resource owner
    - User
  - Client
    - 3rd party application
  - Authorization server
    - The one who issues tokens

OAuth2

- Abstract protocol flow



OAuth2

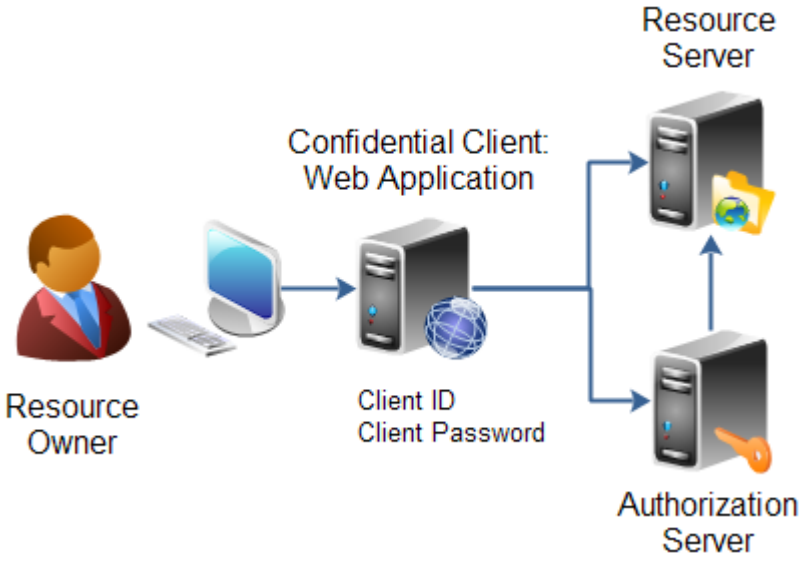
- Client types and profiles
  - We consider 2 types of clients
    - Confidential
      - Take place if client secret is known only for client application
        - Especially is not shared with resource owner
    - Public
      - The opposite situation

OAuth2

- Client types and profiles
  - Protocol emphasizes 3 types of clients
    - Server-side web application
    - Client-side application running in a web browser
    - Native application

OAuth2

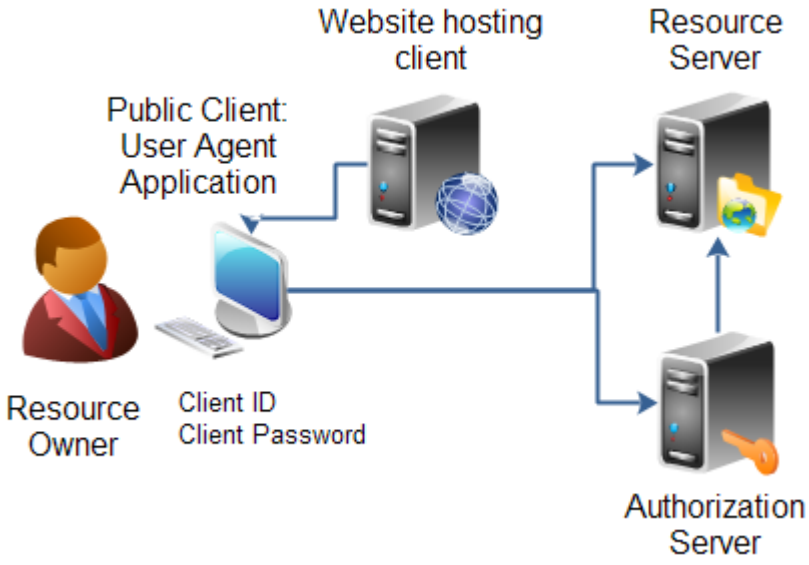
- Client types and profiles
  - Protocol emphasizes 3 types of clients
    - Server-side web application
      - The application makes API calls using a server-side programming language
      - The user has no access to the OAuth client secret or any access tokens issued by the authorization server



Source: http://tutorials.jenkov.com/oauth2/client-types.html

OAuth2

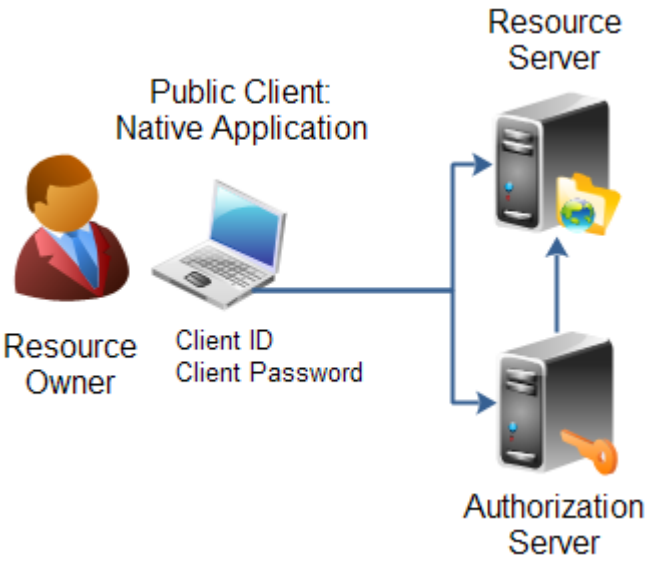
- Client types and profiles
  - Client-side application running in a web browser
    - The application makes API calls form web browser technology like JavaScript or Flash
    - Usually it is a SPA-like app hosted on web server, but run fully in a web browser



Source: http://tutorials.jenkov.com/oauth2/client-types.html

OAuth2

- Client types and profiles
  - Native application
    - Similar solution as client-side application
    - Usually it is desktop or mobile application
    - Difference is that everything is stored on user's device



Source: http://tutorials.jenkov.com/oauth2/client-types.html

OAuth2

- Key characteristics
  - OAuth2 is about authorization
    - Or, actually, it's a delegation protocol
  - Access is granted based on access_token
    - which doesn't include anything about identity
  - Resource owner agrees to share resources with a third party application
    - So called „consent screen"
  - Client doesn't get the user's password in code and implicit flows

OAuth2

- Registration
  - Usually in real world, an application (client) needs to register in the resource server
  - On other words, there is a trust between client and resource server, client authenticates in RS
  - As a outcome, usually client gets
    - Client ID
    - Client Secret
      - Client secret is required only in the confidential clients flows
  - Additionally with client application a redirect URI is associated
    - Used when user (resource owner) successfully authenticates on authorization server

OAuth2

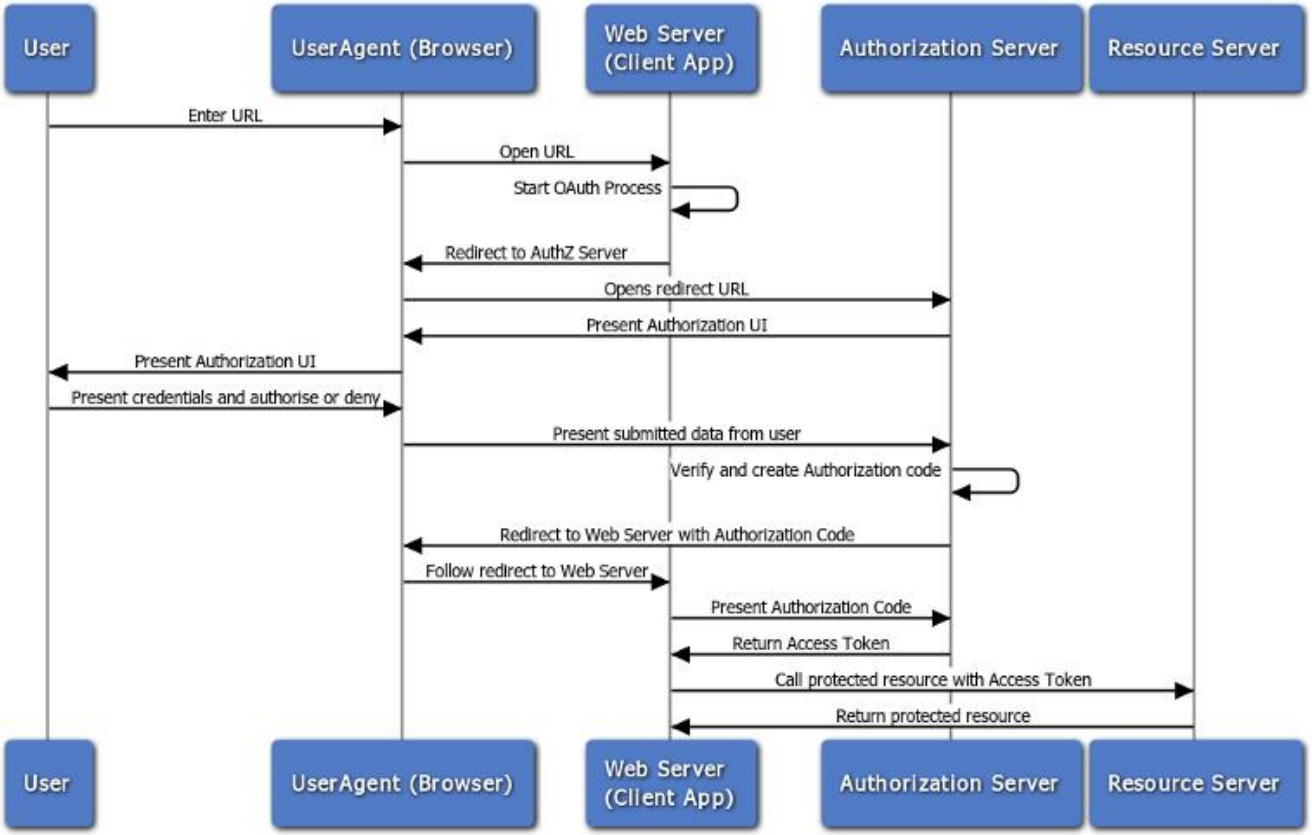
- Authorization flows
  - Authorization Code Flow
  - Implicit Flow
  - Resource Owner Credential Flow
  - Client Credential Flow

OAuth2

- Authorization Code Flow
  - Dedicated for web applications
  - Client can store secret securely on the server
  - Access token never sent to a browser
    - Browser gets a code exchanged later for an access token
  - Tokens
    - Access token: short time, gives access to the resource
    - Refresh token: long time, allows to get a new access token
  - This is most often used flow

OAuth2

- Authorization Code Flow



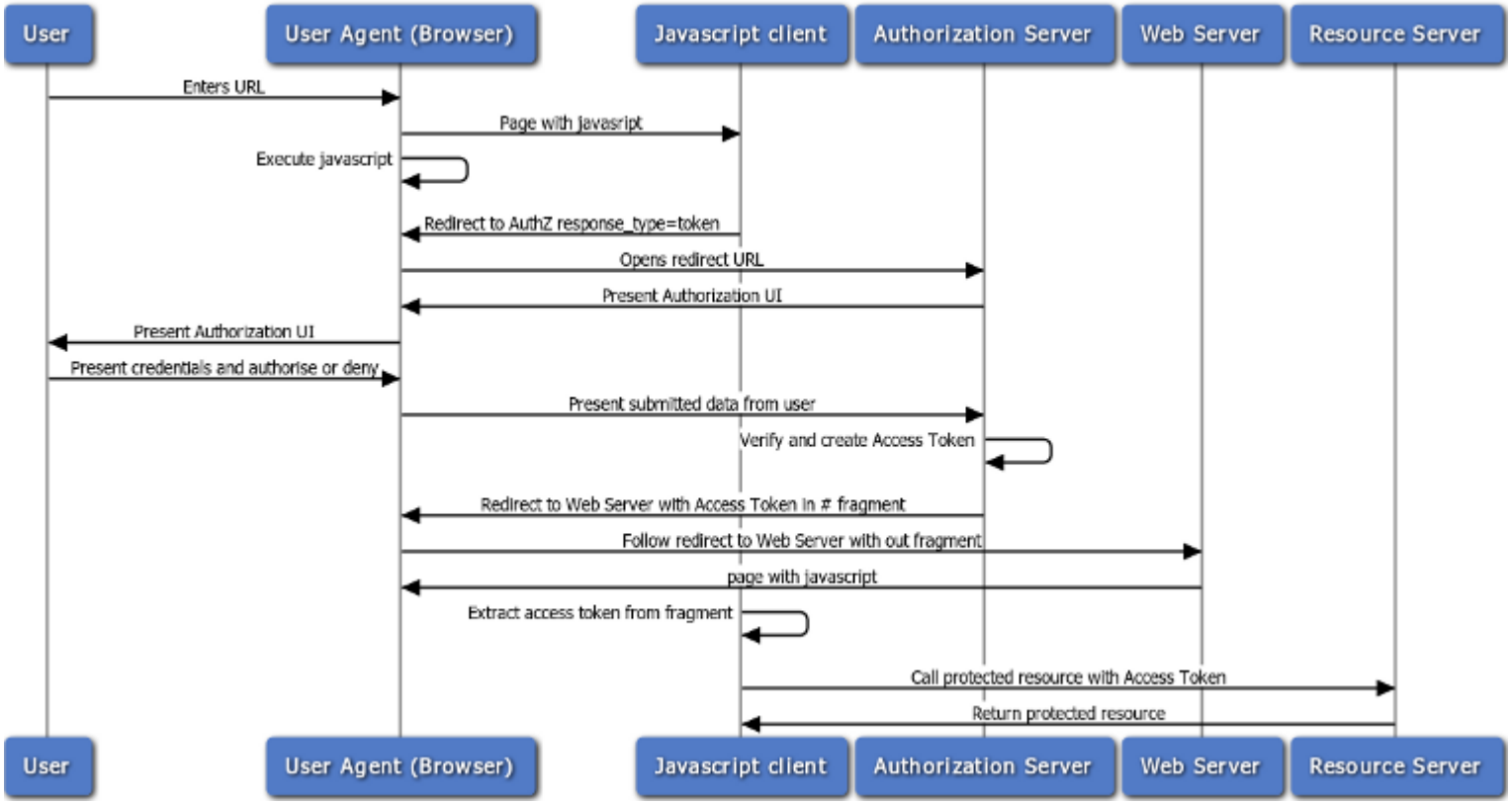
https://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/oauth_flows.html

OAuth2

- Implicit Flow
  - Dedicated to desktop, SPA and mobile applications
  - Very similar to code flow, but there is no code, access token is sent directly to device
  - There is no refresh token

OAuth2

- Implicit Flow



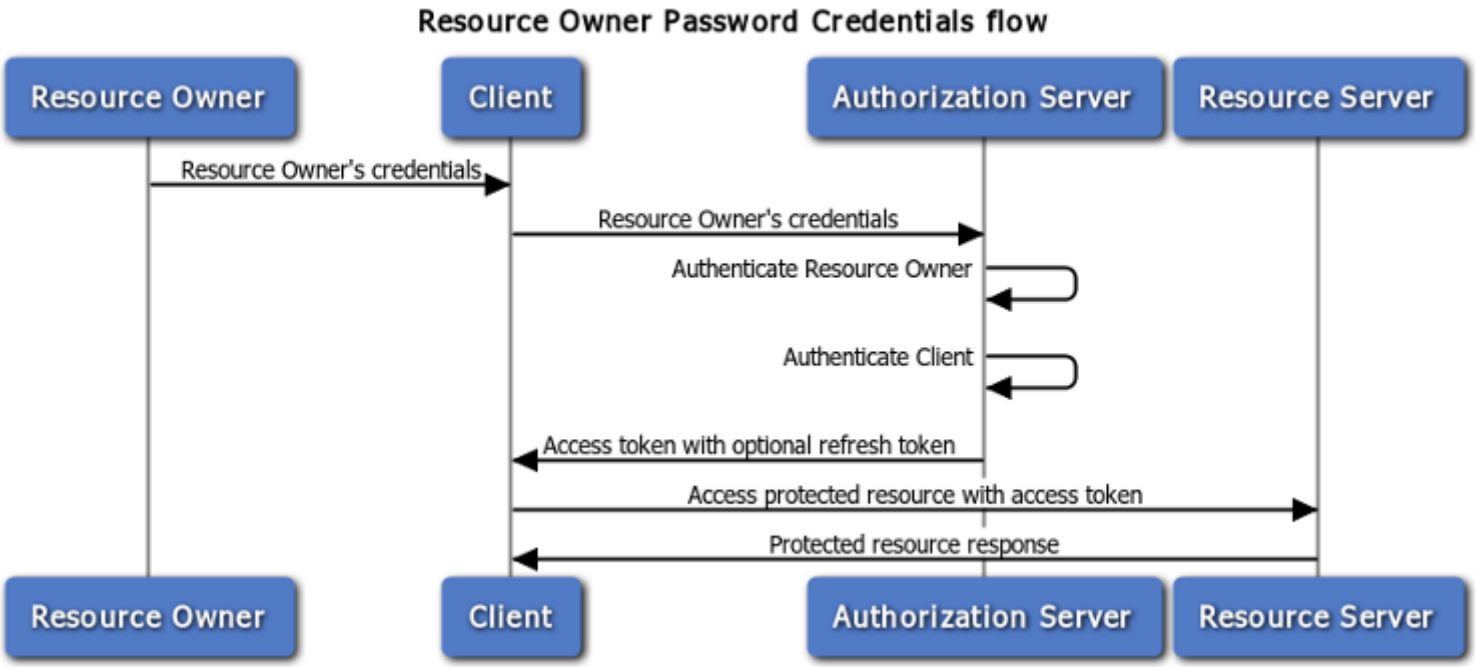
https://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/oauth_flows.html

OAuth2

- Resource Owner Credential Flow
  - In previous flows authentication is performed on AS
  - In this case client directly authenticate on AS
    - Client get the username and password and use it for authentication
    - Client should forget the password after authentication
      - What means, that client application must be trusted
  - Authorization response
    - with access & refresh token
  - Client app use access token to access resources

OAuth2

- Resource Owner Credential Flow



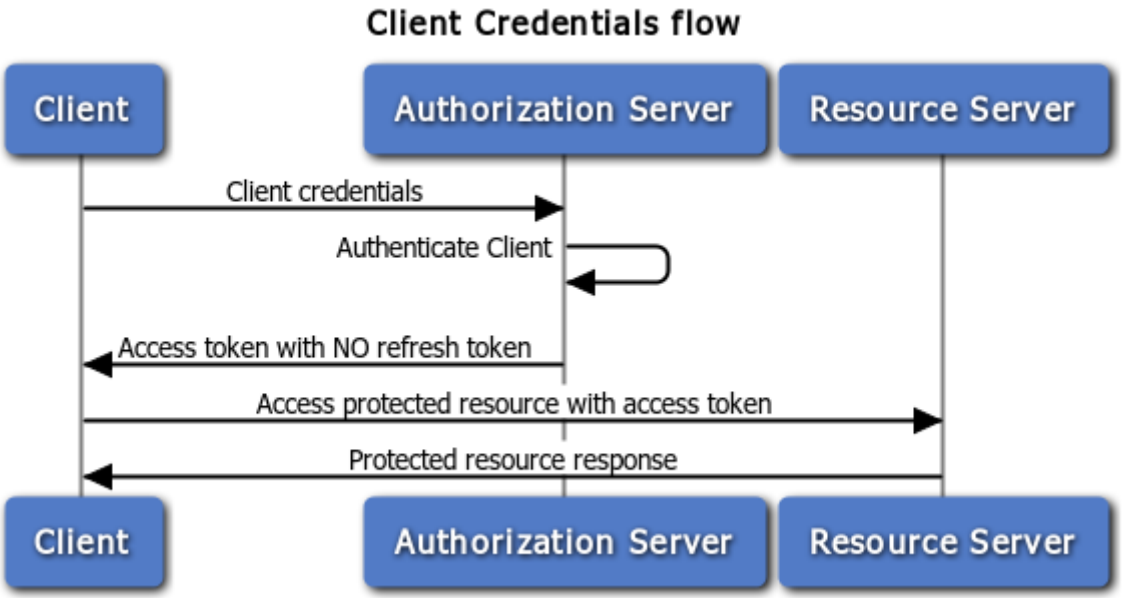
https://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/oauth_flows.html

OAuth2

- Client Credential Flow
  - Use for "service to service" communication
  - Client application itself ask AS for token
  - Client apps doesn't do this "on behalf" of some user – there is no user involved.
  - Provided are client_id and client_secret
    - See example: http://dev.mendeley.com/reference/topics/authorization_client_credentials.html

OAuth2

- Client Credential Flow



https://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/oauth_flows.html

OAuth2

- Summary of use cases
  - Web-server applications
    - Authorization code flow
  - Browser based applications
    - Implicit flow
  - Username/password access
    - Resource Owner Credential Flow
  - Mobile applications
    - Implicit flow
  - Application access
    - Client credentials flow

OAuth2

- Why authorization grant flow? Why not just implicit one?
  - No need to HTTPS between browser and client
  - No risk that JavaScript (or whatever else) steal an access token
  - Client Secret prevents from malicious app getting access to data (e.g. by poisoning the DNS)
- On the other hand it is important that in implicit flow access token is only on client
  - No risk that any other party of solution with steal an access token
  - No risk of man-in-the-middle
    - The relations is only between client and resource server

http://stackoverflow.com/questions/13387698/why-is-there-an-authorization-code-flow-in-oauth2-when-implicit-flow-works-s
https://salesforce.stackexchange.com/questions/14009/whats-the-benefit-of-the-client-secret-in-oauth2

OAuth2

- Scopes
  - Define what authorizarions will be given
  - Are expressed as a set of case-sensitive and space-delimited strings
  - Should be shared between auth and resource
  - Example scopes:
    - Google:
      - https://www.googleapis.com/auth/analytics
      - https://www.googleapis.com/auth/calendar
      - https://www.googleapis.com/auth/gmail.readonly
    - Facebook: user_friends, email, user_photos, user_posts

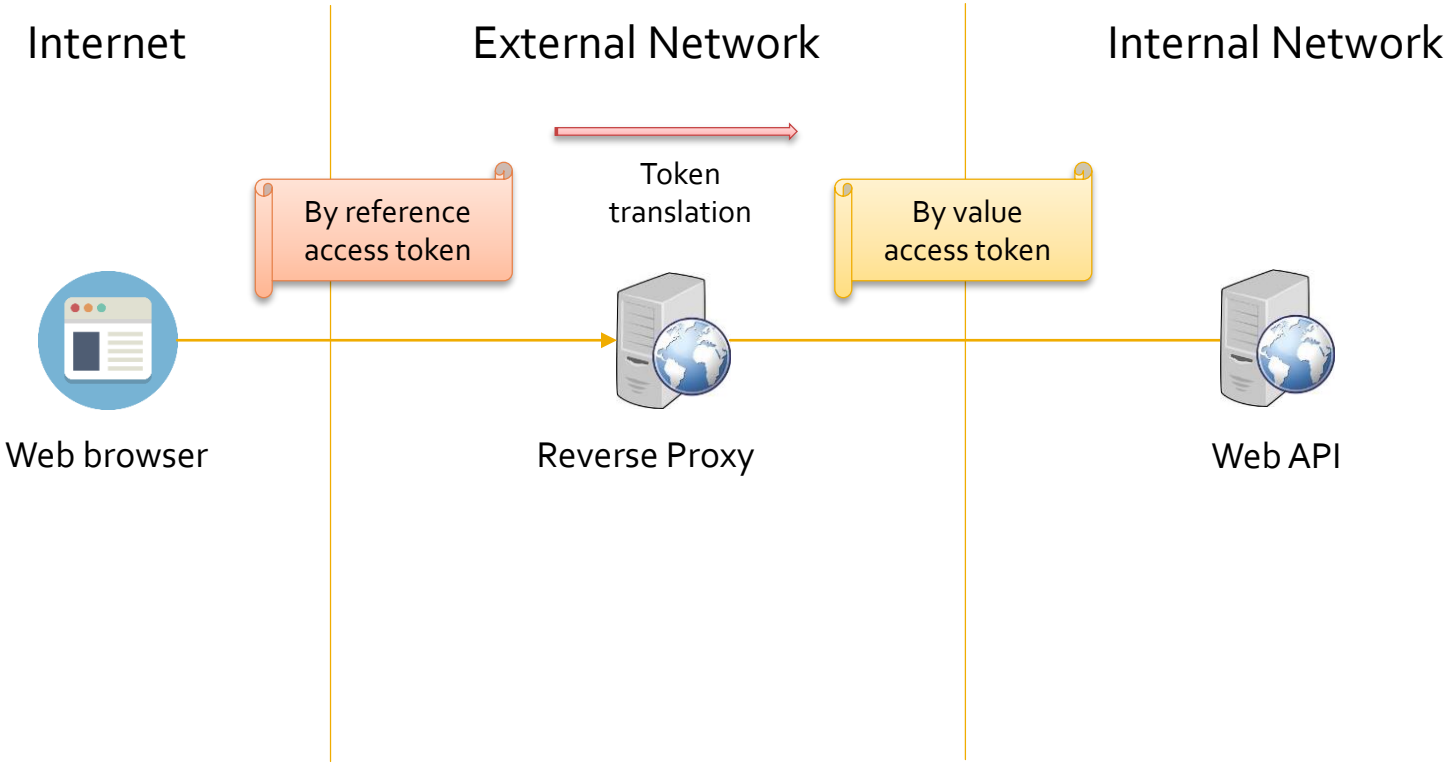
https://developers.google.com/identity/protocols/googlescopes
https://developers.facebook.com/docs/facebook-login/permissions/

OAuth2

- What is access token?
  - There are 2 types of tokens
    - By value or self-contained, e.g. JWT
    - By reference, e.g. random string
  - Which approach is better?
    - JWT
      - no need to ask AS, but no way to revoke
        - problem with long lived AT, logout operations
    - By reference
      - Central management, but there is a need for additional communication is AS

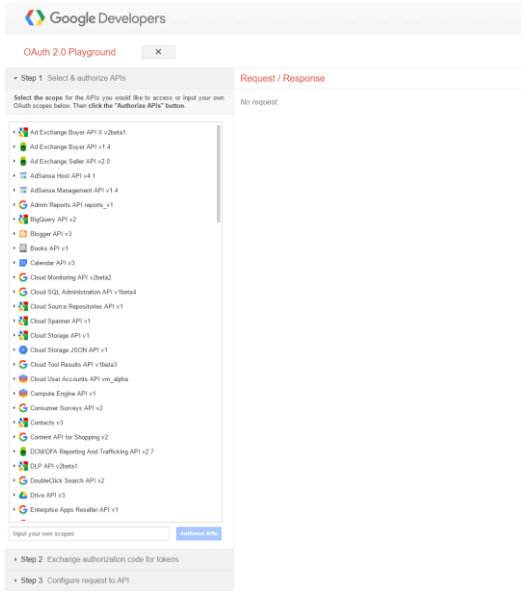
OAuth2

- Tokens in OAuth2, balanced approach



OAuth2

- Let's play
  - https://developers.google.com/oauthplayground/



References

- Tokens consideration
  - https://autho.com/blog/2014/01/07/angularjs-authentication-with-cookies-vs-token/
  - https://autho.com/blog/2014/01/27/ten-things-you-should-know-about-tokens-and-cookies/
  - http://jpadilla.com/post/73791304724/auth-with-json-web-tokens
  - http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html
  - https://developer.atlassian.com/static/connect/docs/concepts/understanding-jwt.html
  - http://msdn.microsoft.com/en-us/library/gg185950.aspx
  - http://www.slideshare.net/briandavidcampbell/owasp-vancouver
  - http://stackoverflow.com/questions/18677837/decoding-and-verifying-jwt-token-using-system-identitymodel-tokens-jwt
  - http://dotnetcodr.com/2014/01/20/introduction-to-oauth2-json-web-tokens/
- OAuth2 by Oracle
  - https://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/oauth_flows.html
- OAuth2 Threat model and security consideration
  - https://tools.ietf.org/html/rfc6819