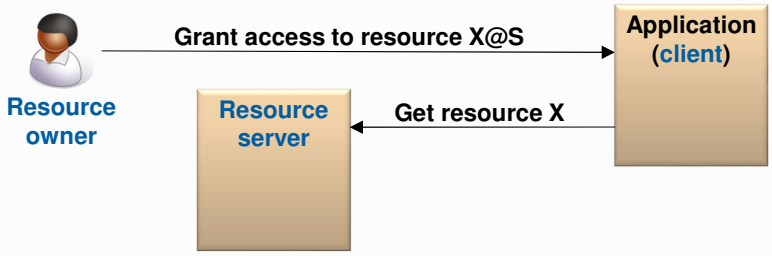
OAuth 2.0 authorization framework

Goal

- Allow an application to access user resources maintained by a service/server



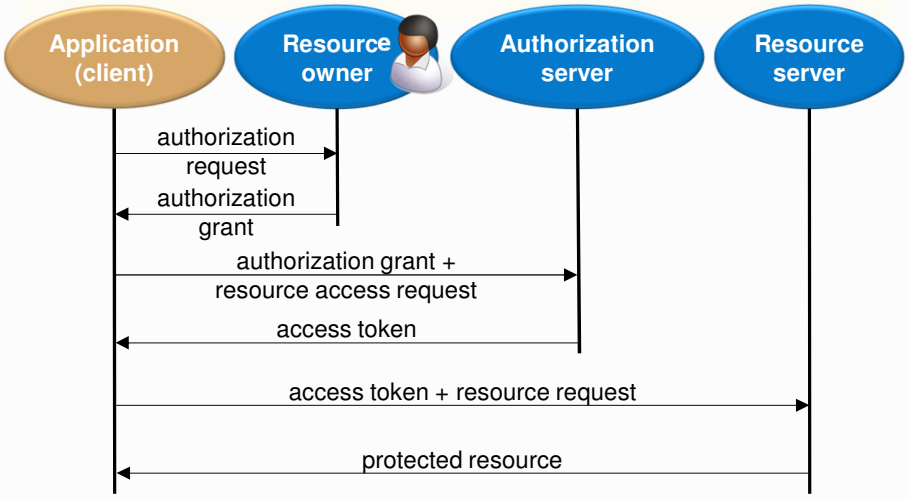
Roles (RFC 6749)

- Resource owner
  - An entity capable of **granting access** to a **protected resource**
  - **End-user**: a resource owner that is a person
- Resource server
  - The server hosting protected resources
  - Capable of accepting and responding to protected resource requests using **access tokens**

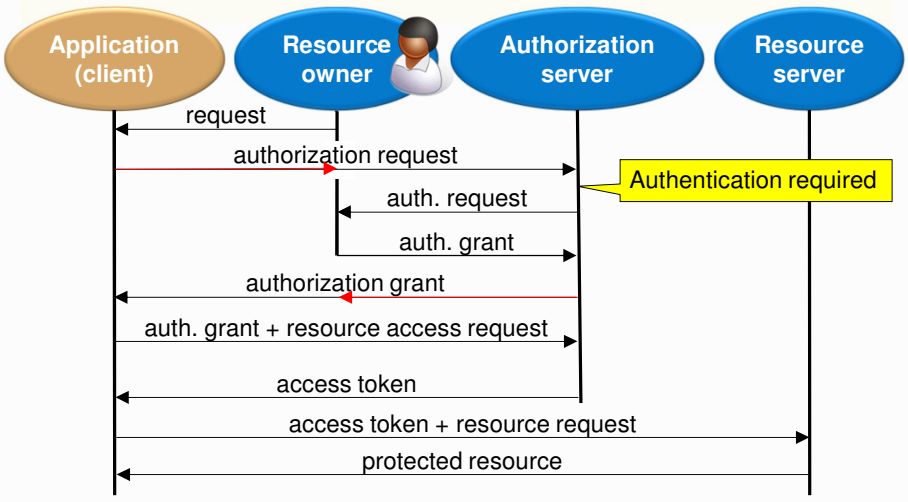
Roles (RFC 6749)

- Client
  - An **application** making requests for protected resources on behalf of the resource owner and with its authorization
- Authorization server (aka OAuth server or provider)
  - The server issuing **access tokens** to the client after successfully **authenticating** the resource owner and obtaining its **authorization** for the client to access one of its resources

Abstract protocol flow (RFC 6749)



Actual protocol flow



Communication endpoints: Authorization endpoint

- Service provided by the OAuth server
  - Authenticates the resource owner
  - Asks for the delegation of access rights to its protected resources to the client
  - Send an authorization grant to the redirection endpoint

Communication endpoints: Token endpoint

- Service provided by the OAuth server
  - Produces access tokens given an authorization grant
  - It can also produce refresh tokens
  - Refresh tokens can be used to get new tokens
    - With an authorization grant
- Client authentication
  - ClientID + ClientSecret + HTTP basic authentication

Communication endpoints: Redirect endpoint

- Service provided by the client
  - It collects the authorization grant provided by the OAuth server
  - It should be called by the OAuth server using an HTTP redirect

Application (client) types

- Type is related with the ability to maintain the confidentiality of client credentials
  - Even from the resource owner
- Confidential
  - Capable
  - e.g. a secure server
- Public
  - Incapable
  - e.g. a web browser-based application, a mobile App

Application (client) profiles

- Web application
  - Confidential client running on a web server
- User-agent based application
  - Public client where the client code runs on a user-agent application (e.g. a browser)
- Native application
  - Public client installed and executed on the device used by the resource owner

Application (client) registration (in an OAuth server)

- Clients accessing OAuth servers must be previously registered
  - Nevertheless, the standard does not exclude unregistered clients
  - A registered client is given a unique identifier
    - ClientID
- Registration includes both informational, legal and operational information
  - Redirection URLs
  - Acceptance of legal terms
  - Application (client) name, logo, web site, description
  - Client type
  - Client authentication method (for confidential clients)

OAuth tokens: Authorization grant

- Created by an OAuth server
  - Upon authenticating a resource owner and getting its consent to grant access to a protected resource
  - An opaque byte blob that makes sense only to its issuer
- Short validity time
  - Just enough to get an access token

OAuth tokens: Access token

- Created by an OAuth server
  - Upon authenticating a client and receiving an authorization grant
  - An opaque byte blob that makes sense to its issuer and to the resource owner
    - An access capability
- Bearer tokens
  - Clients need to protect their use with HTTPS
  - Clients can handover tokens to others

OAuth tokens: Refresh token

- Created by an OAuth server
  - When creating an access token
  - An opaque byte blob that makes sense only to its issuer
  - It can be used to collect a new access token
    - Still requiring the client authentication
- Bearer tokens
  - Clients need to protect their use with HTTPS
  - Clients can handover tokens to others

OAuth flows

- Authorization code flow
  - 3-legged OAuth
  - Default OAuth flow
  - The most secure
- Implicit flow
- Resource owner password credentials flow
- Client credentials flow
  - 2-legged flow

Authorization code flow

- 3-legged OAuth
  - It enables checking the identity of the three involved actors
- OAuth server authenticates the resource owner
  - Username + password or other means
- OAuth server authenticates the client
  - ClientID + ClientSecret + HTTP basic authorization
- Client authenticates the OAuth server
  - Certificate + URL

Authorization code flow

- Requirements
  - Confidential application types
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth

Authorization code flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server redirects the client to the OAuth server
- The OAuth server authenticates the resource owner
  - And sends an authorization grant to the client
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
  - Using its authorization grant
- The client uses again the resource server API to get a resource
  - This time providing an access token

Implicit flow

- Requirements
  - Public application types
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID
    - Not regulated by OAuth
- Limitations
  - No client authentication
  - No refresh tokens

Implicit flow

- Resource owner uses a mobile or client-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server redirects the client to the OAuth server
- The OAuth server authenticates the resource owner
  - And sends an access token to the client
- The client uses again the resource server API to get a resource
  - This time providing an access token

Resource owner password flow

- Requirements
  - Confidential application types
  - Sharing of resource owner credentials with client applications
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth
- Limitations
  - Resource owners need to trust on client applications

Resource owner password flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server requests a token
- The client asks the resource owner for authentication credentials
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
  - Using the resource owner's credentials
  - These should be immediately discarded
- The client uses again the resource server API to get a resource
  - This time providing an access token

Client credentials flow

- Requirements
  - Confidential application types
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth
- Limitations
  - No resource owner authentications or authorizations

Client credentials flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server requests a token
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
- The client uses again the resource server API to get a resource
  - This time providing an access token