Introduction to REST Endpoint Security

OAuth2

Java Implementations

Lesson Objectives

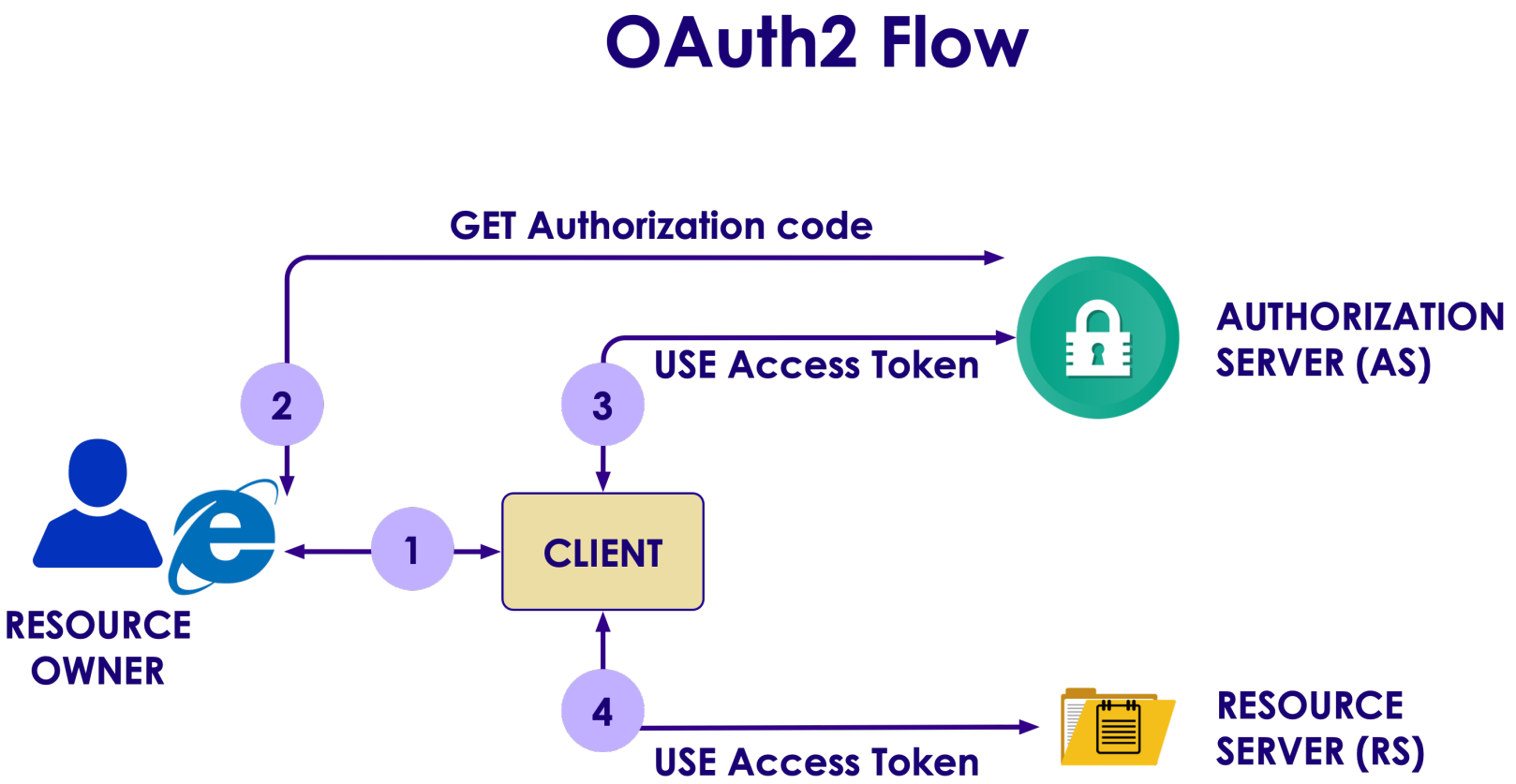
- Understand the needs that OAuth2 addresses
- Be familiar with OAuth2 capabilities and advantages
- Gain an understanding of a basic OAuth2 Implementation

OAuth2

OAuth2

Java Implementations

What Is It?



Why OAuth?

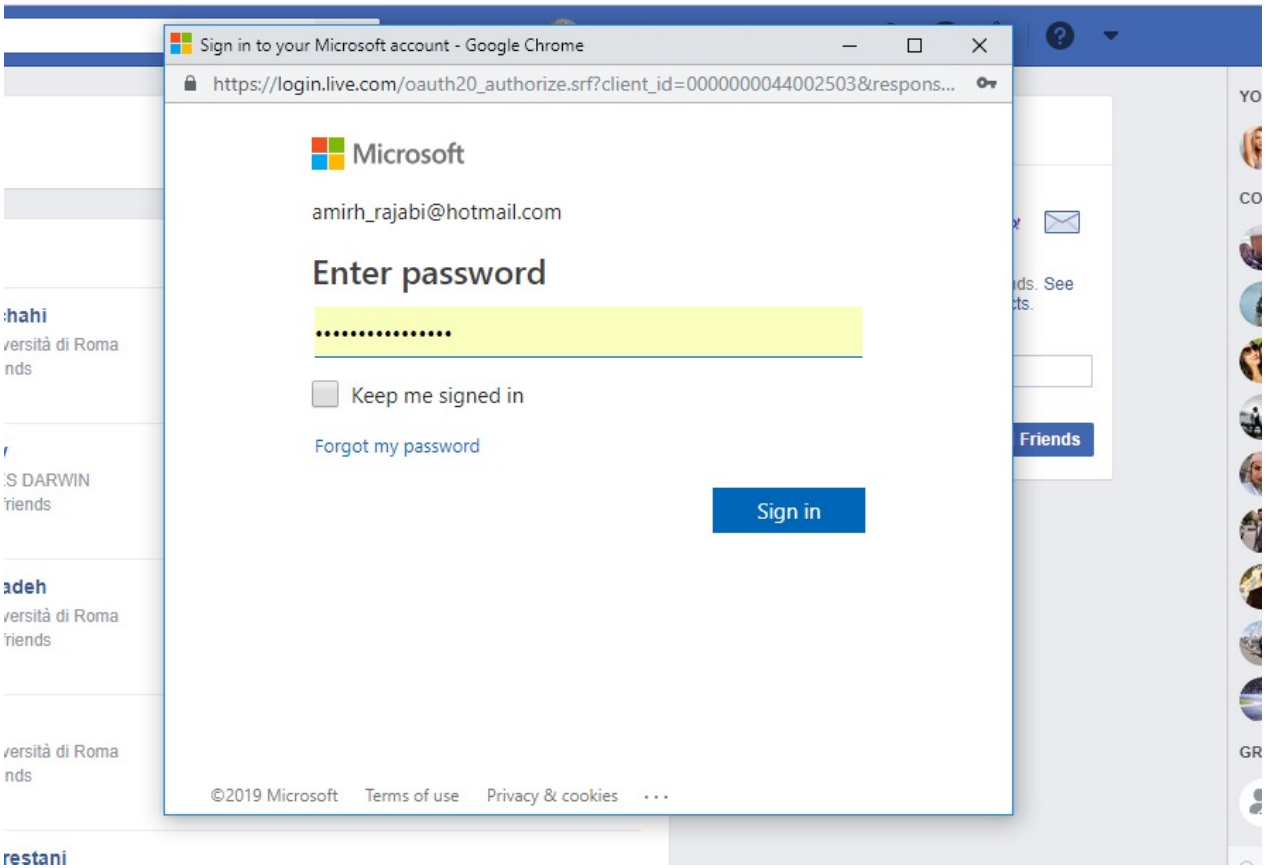
- Third party applications can access user's resources without knowing their credentials
- Limits access to HTTP services
- Softwares and packages don't store user's credentials anymore (only tokens)
- Based on TLS/SSL
- No backward compatibility
- Easily revokable

A Little Bit History

- OAuth 1.0
- Core specification - 2007
- OAuth 1.0a
- A security issue was fixed - 2009
- OAuth 2.0
- Standardized - 2012
- More security and simplicity

A Good Use Case

- In some websites you can invite your friends by importing your contant list
- Look at the address bar:



Roles

- Resource owner
- Resource server
- Client
- Authorization server

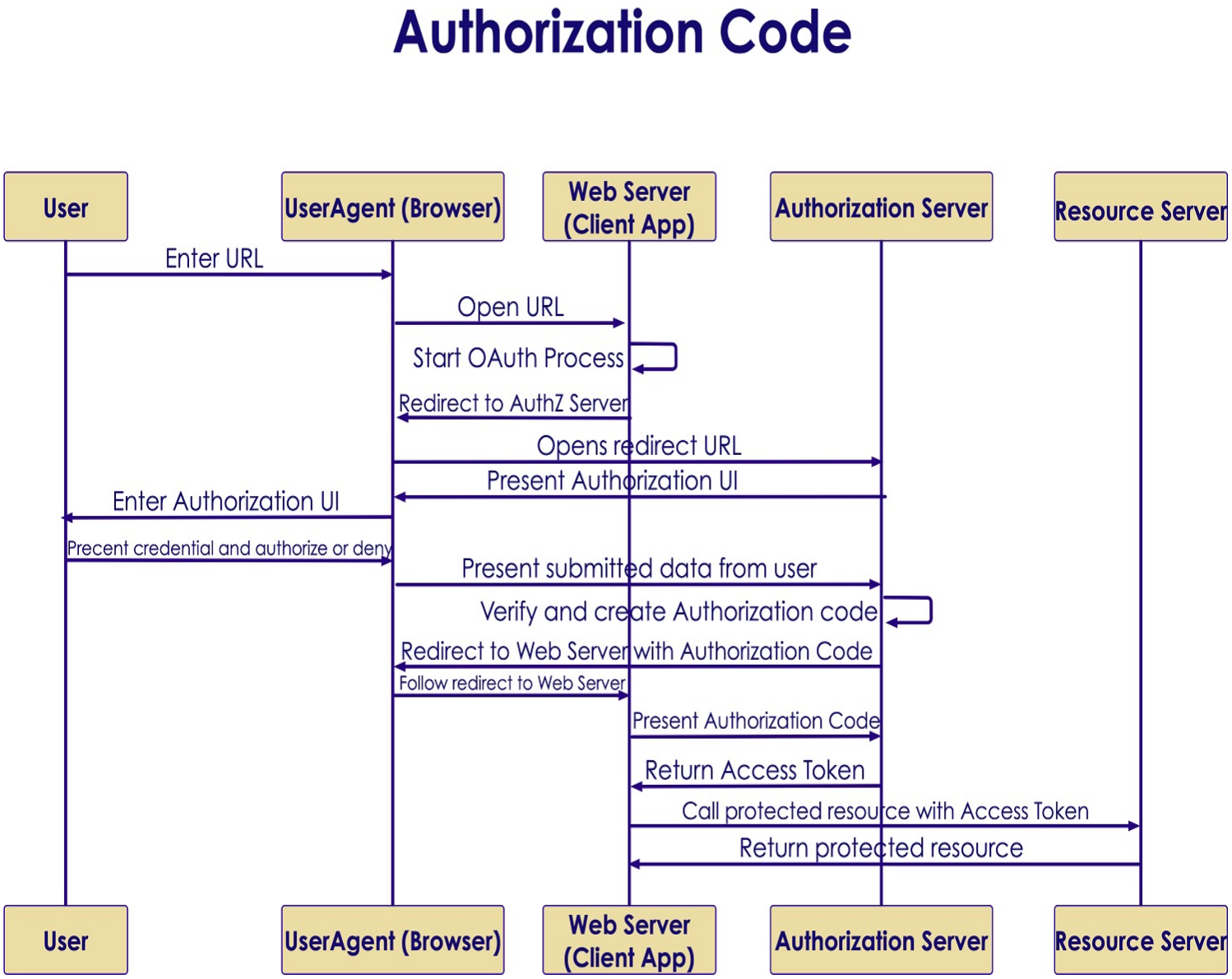
Grant Types

- Authorization code (web applications)
- Implicit (mobile and browser-based applications)
- Resource owner password credentials (user + password)
- Client credentials (application)
- Device Code (electronic device)

Authorization Code

- This grant type is used by confidential and public clients
- Exchanges an authorization code for an access token
- User returns to the client through redirect URL
- Then application gets the authorization code from URL
- Then use it to request an access token

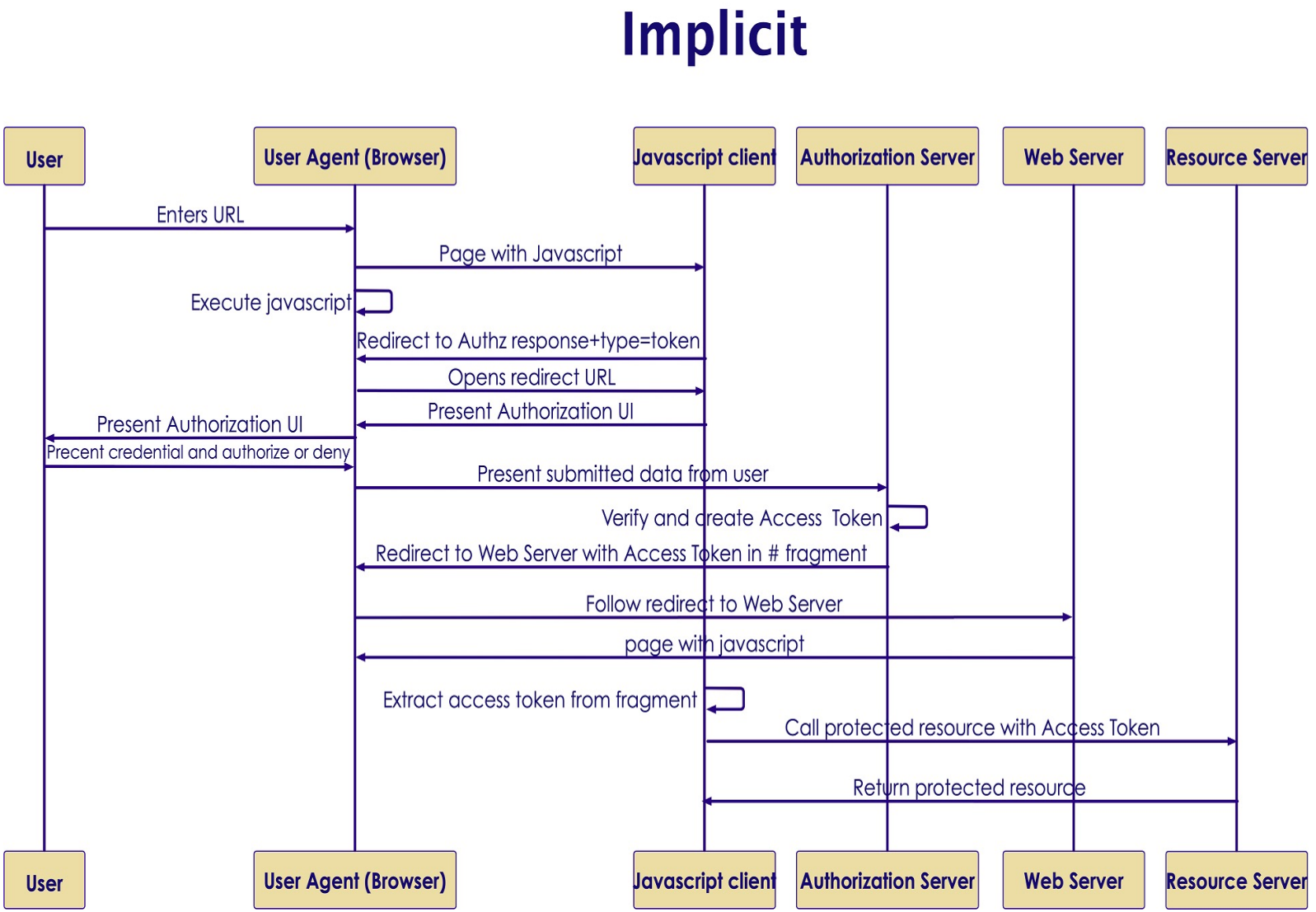
Authorization Code In A Picture



Implicit

- Used by public clients
- Access token is returned without an extra authorization code exchange
- Some servers ban this flow
- It is not recommended

Implicit In A Picture

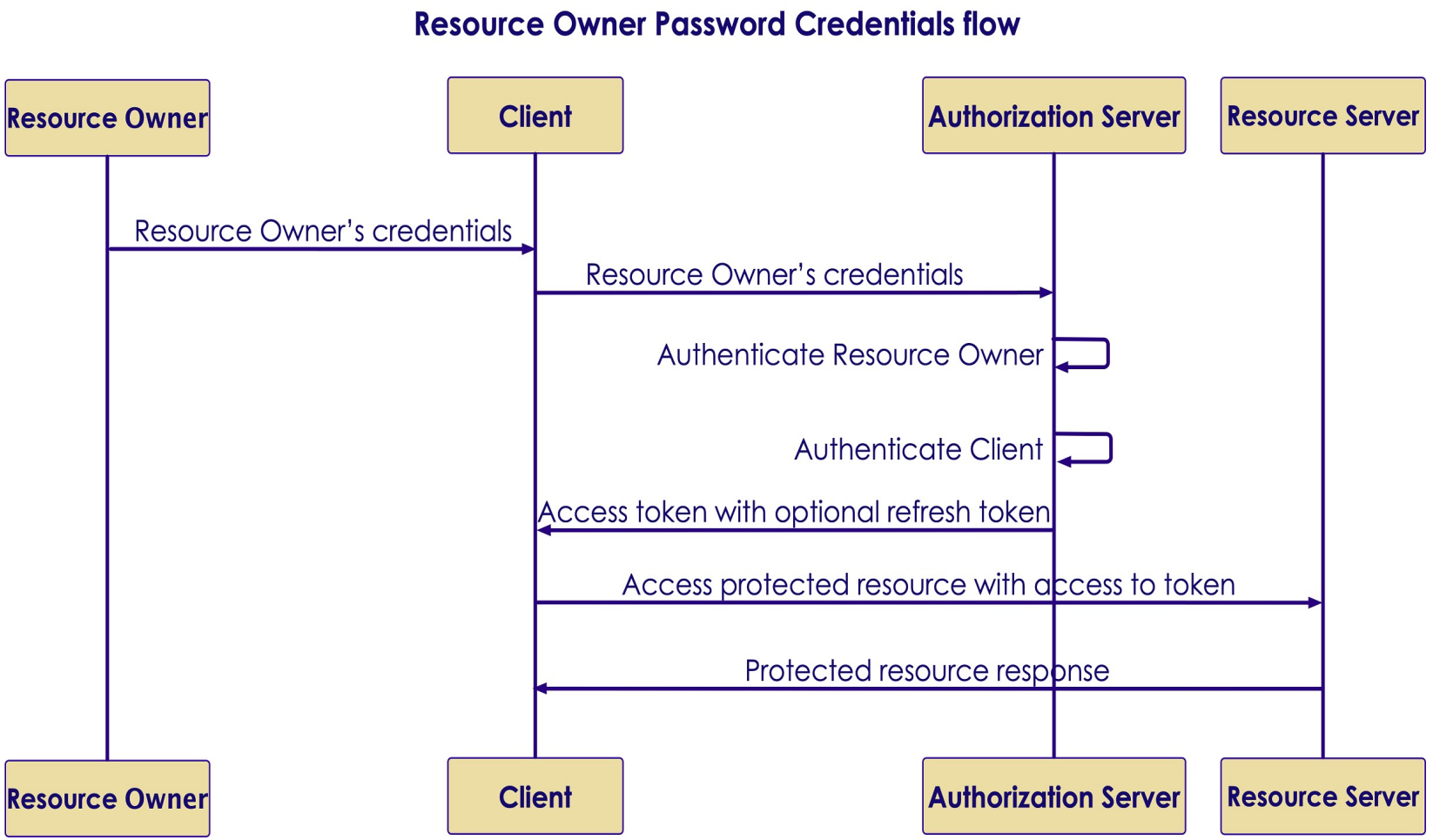


Resource Owner Password Credentials

- First-party clients use this type to exchange user's credential for a token
- Asks the user for their credential
- Don't let third party clients to use it
- Username and password are exchanged directly for a token

Password Credentials In A Picture

Resource Owner Password Credentials

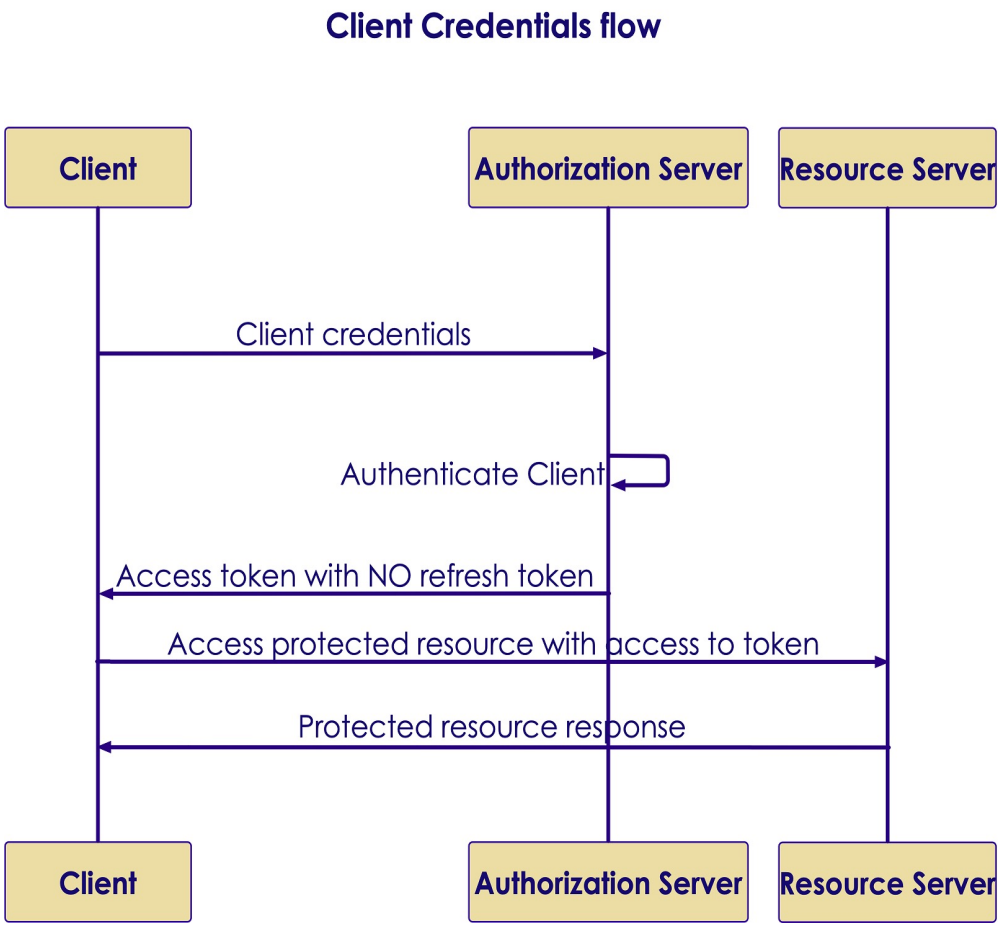


Client credentials

- Is used by clients to obtain a token out of the user's context
- To access resources about themselves instead of accessing a user's resource

Client credentials In A Picture

Client Credentials



Device Code

- Is used by browserless or input-constrained devices
- To exchange a previously obtained device code for an access token
- Value: `urn:ietf:params:oauth:grant-type:device_code`

Tokens

- Types according to properties
  - Bearer
    - Large random
    - Uses SSL to protect
    - It is stored as a hash on the server
  - Mac (Not Recommended)
    - Uses a nonce to prevent replay
    - Does not use SSL
    - OAuth 1.0 only supported
- Types according the life cycle
  - Access token
    - Short
  - Refresh token
    - Long

Bearer Token

- Predominant type of access token
- An opaque string, without any meaning
- Some servers issue short string and some issue `JSON Web Tokens`

Pros and Cons of OAuth2

- Pros
  - Enables integration of third party applications to websites
  - Enables granting limited access either scope or duration
  - User does not have to enter password on third party site
- Cons
  - Complexity in development of authorization server
  - Compatibility issues

Java Implementations

OAuth2

Java Implementations

Some Java Implementations

- Jersey
- Apache oltu
- Spring security (Pretty Popular)

Jersey

- It is an Open source RESTful web services framework
- Supports and extends JAX-RS API and extends
- Integrates with the Java EE standard security
  - `@RolesAllowed`
  - `@PermitAll`
  - `@DenyAll`
- Supports entity filtering
  - `@EntityFiltering`
- Only supports OAuth2 at client side

Goals of Jersey Project

- Track the JAX-RS API and provide regular releases of production quality Reference Implementations that ships with GlassFish
- Provide APIs to extend Jersey & Build a community of users and developers; and finally
- Make it easy to build RESTful Web services utilising Java and the Java Virtual Machine

Java EE security integration

```
1  @Path("restricted-resource")
2  @Produces("application/json")
3  public class restricted_resource {
4    @GET @Path("denyAll")
5    @DenyAll
6    public restricted_entity denyAll() {...}
7
8    @GET @Path("rolesAllowed")
9    @RolesAllowed({"manager"})
10   public restricted_entity rolesAllowed() {...}
11 }
```

Client support

```
1  OAuth2CodeGrantFlow.Builder builder =
2    OAuth2ClientSupport
3      .authorizationCodeGrantFlowBuilder(
4        clientId,
5        "https://example.com/oauth/authorization",
6        "https://example.com/oauth/token"
7      );
8
9  OAuth2CodeGrantFlow flow = builder.property(
10   OAuth2CodeGrantFlow.Phase.AUTHORIZATION,
11     "readOnly","true")
12     .scope("contact")
13     .build();
14
15 String authorizationUri = flow.start();
16 ...
17 final TokenResult result = flow.finish(code,state);
18 ...
```

Apache Oltu

- Apache OAuth protocol implementation
- Also covers other implementations
  - JSON Web Token (JWT)
  - JSON Web Signature (JWS)
  - OpenID connect
- Supports OAuth2 features completely
  - Authorization server
  - Resource server
  - Client
- Provides predefined OAuth2 client types
  - `Github`, `Facebook`, `Google`, etc

Authorization endpoint

```
protected void do_get(HttpServletRequest req,
        HttpServletResponse resp)
            throws ServletException, IOException {

      //dynamically recognize an OAuth profile and perform validation
      OAuthAutzhRequest oauth_req = new OAuthAuthzRequest(req);
      validateRedirectionURI(oauth_req)

      //build OAuth response
      OAuthResponse resp = OAuthASResponse
        .authorizationResponse(HTTPServletResponse.SC_FOUND)
        .setCode(oauthIssuerImpl.authorizationCode())
        .location(ex.getRedirectUri())
        .buildQueryMessage();

      resp.sendRedirect(resp.getLocationUri());
    }
```

Token endpoint

```
protected void do_post(HttpServletRequest req,
        HttpServletResponse resp)
            throws ServletException, IOException {

            OAuthIssuer oauthIssuerImpl =
                new OAuthIssuerImpl (new MD5Generator());

            OAuthTokenRequest oauth_req =
                new OAuthTokenRequest (req);

            validateClient(oauth_req);

            String authz_code = oauth_req.getCode();
            String access_token = oauthIssuerImpl.accessToken();
            String refresh_token = oauthIssuerImpl.refreshToken();

            OAuthResponse r = OAuthASResponse(...);
             }
```

Protecting resources

```
1  Protected void do_get(HttpServletRequest req,
2     HttpServleResponse resp)
3        throws ServletException, IOException {
4
5          //OAuth request and validation
6      OAuthAccessResourceRequest oauth_req = new
7        OAuthAccessResourceRequest (req,
8            ParameterStyle.BODY);
9
10         //Getting access token
11         String accessToken =
12            oauth_req.getAccessToken();
13
14         // ... validate access token
15       }
```

OAuth2 client

```
1  OAuthClientRequest req = OAuthClientRequest
2    .tokenProvider(OAuthProviderType.FACEBOOK)
3    .setGrantType(GrantType.AUTHORIZATION_CODE)
4    .setClientId("your-facebook-client-id")
5    .setClientSecret("your-facebook-client-secret")
6    .setRedirectURI("http://www.mysite.com/redirect")
7    .setCode(code)
8    .buildQueryMessage();
9
10 //create OAuth client that uses custom http client under the hood
11 OAuthClient oauth_client = new OAuthClient(new URLConnectionClient());
12 OAuthAccessTokenResponse oauth_resp = oauth_client.accessToken(req);
13 String access_token = oauth_resp.getAccessToken();
14 String expires_in = oauth_resp.getExpiresIn();
```

Spring security OAuth

- Supports OAuth (1a) and OAuth2
- Implements 4 types of authorization grants
- Supports all OAuth2 features:
  - Authorization server
  - Resources server
  - Client
- Good integration with JAX-RS and Spring MVC
- Configuration using annotation support
- Integrates with the `Spring` platform

Authorization server

- `@EnableAuthorizationServer`
  - For configuring OAuth2 authorization server
  - XML configuration related: `<authorization-server/>`
- `ClientDetailsServiceConfigurer`
  - Defines the client details service
  - In-memory or JDBC implementation
- `AuthorizationServerTokenServices`
  - Operations to manage OAuth2 tokens
  - Tokens in-memory, JDBC or JSON Web Token (JWT)
- `AuthorizationServerEndpointConfigurer`
  - Supports grant types
  - Password types not supported

Resource server

- can be the same as Authorization server or in a separate application
- Authentication filter for web protection
- `@EnableResourceServer`
  - For configuring OAuth2 resource server
  - XML config `<resource-server/>` *Supports expresson-based access control
  - `#oauth2.clientHasRole`
  - `#oauth2.clientHasAnyRole`
  - `#oauth2.denyClient`

Client

- Storing the current request and context by creating filter
- Manages:
  - redirection to OAuth
  - redirection from OAuth
- `@EnableOAuth2Client`
  - To configure OAuth2 client
  - XML config related `<client/>`
- `OAuth2RestTemplate`
  - Wrapper client object to access the resources

Lab

- Spring + REST
- Set up the framework
- Secure it
- The link will be provided