Spring security - II

Teaching Faculty: Dr. Muhyieddin Al-Tarawneh

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Refresh Tokens

- APIs may return two tokens
  - Access token with an expiration time
  - Refresh token with no expiration time

- Refresh token used to get a new access token
  - No additional authentication required

Oauth 2.0

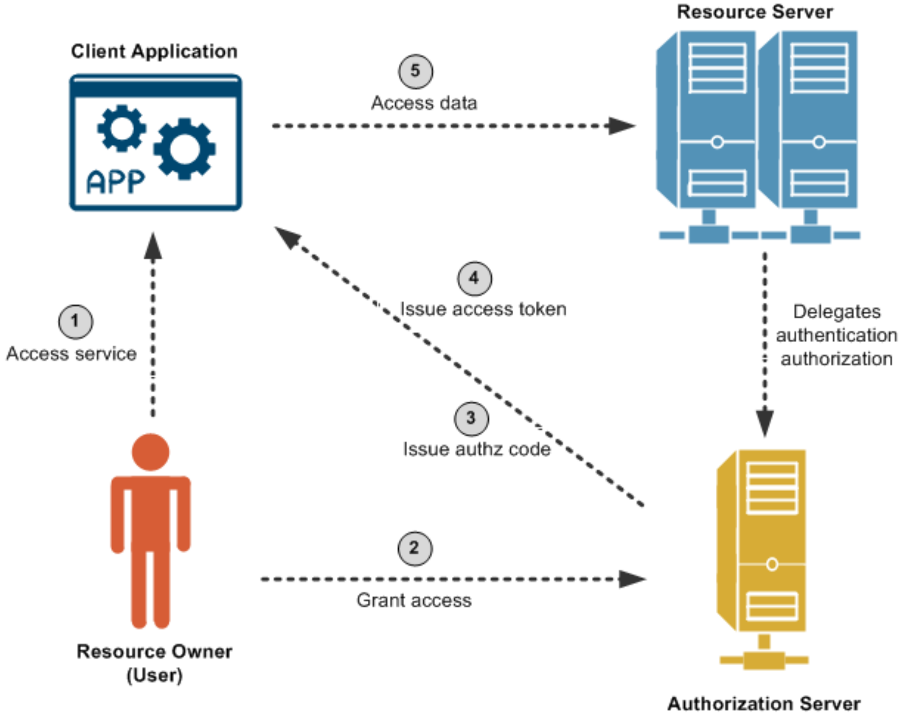
- OAuth 2.0 is the industry-standard protocol for authorization.
- OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.
- It works by delegating user authentication to the service that hosts a user account and authorizing third-party applications to access that user account.

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

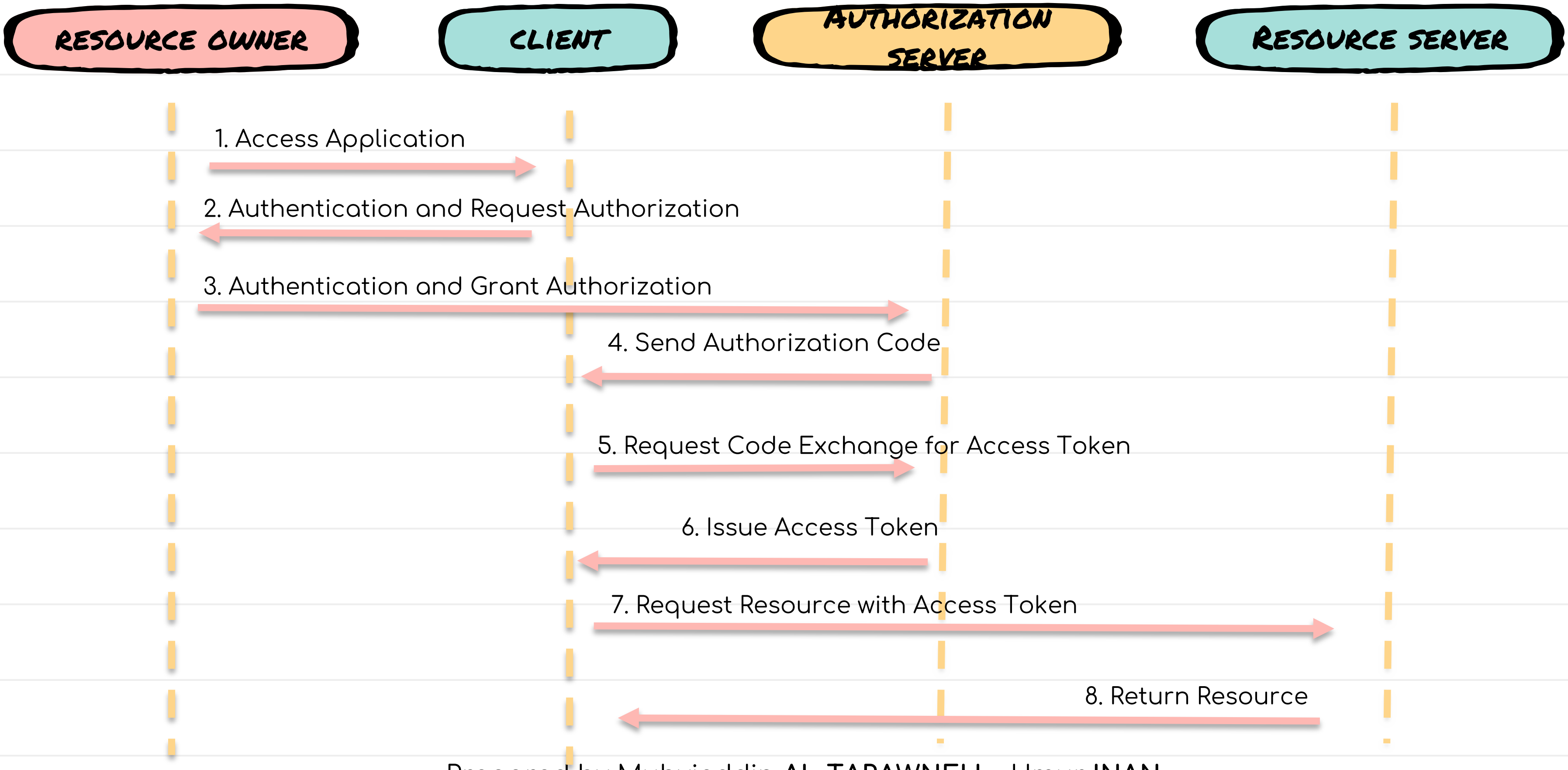
Delegated authorization

- OAuth2 is a standard protocol that solves the delegated authorization problem.
- Users may give permission to "Some App" to access resources on "Another App" so the app can access some resources.
- This is a better way than giving the App our username/password to access resources on our behalf.
- Unfortunately, some apps (Banks, Mint: financial dashboard) are still collecting username/password to access users' data.
- I trust Google but I kind of trust this new App. I want the App to have access to my contacts only.

OAUTH 2.0 FLOW



AUTHORIZATION CODE FLOW



RESOURCE OWNER

CLIENT

AUTHORIZATION SERVER

RESOURCE SERVER

1. Access Application
2. Authentication and Request Authorization
3. Authentication and Grant Authorization
4. Send Authorization Code
5. Request Code Exchange for Access Token
6. Issue Access Token
7. Request Resource with Access Token
8. Return Resource

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Resource Owner

- Normally your application's end user that grants permission to access the resource server with an access token.
- A person or system capable of granting access to a protected resource.

Client

- The application that requests the access token

Resource Server

- Accepts the access token and must verify that it's valid. In this case this is your application.
- The resource server is the OAuth 2.0 term for your API server.
- The resource server handles authenticated requests after the application has obtained an access token.
- Large scale deployments may have more than one resource server.

Authorization Server

- The server that issues the access token.

Authorization Code

- The authorization code is obtained by using an authorization server as an intermediary between the client and resource owner.

- The code itself is obtained from the authorization server where the user gets a chance to see what the information the client is requesting and approve or deny the request.

https://www.oauth.com/oauth2-servers/server-side-apps/authorization-code/

Access Token

- An OAuth Access Token is a string that the OAuth client uses to make requests to the resource server.
- Access tokens do not have to be in any particular format, and in practice, various OAuth servers have chosen many different formats for their access tokens.
- Access tokens may be either "bearer tokens" or "sender-constrained" tokens.

Access Token

- Access tokens must not be read or interpreted by the OAuth client. The OAuth client is not the intended audience of the token.
- Access tokens do not convey user identity or any other information about the user to the OAuth client.
- Access tokens should only be used to make requests to the resource server. Additionally, ID tokens must not be used to make requests to the resource server.

Refresh Token

- An OAuth Refresh Token is a string that the OAuth client can use to get a new access token without the user's interaction.
- A refresh token must not allow the client to gain any access beyond the scope of the original grant.
- The refresh token exists to enable authorization servers to use short lifetimes for access tokens without needing to involve the user when the token expires.

Authorization Code Grant

- The Authorization Code grant type is used by confidential and public clients to exchange an authorization code for an access token.

- It is recommended that all clients use the PKCE extension with this flow as well to provide better security.

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Proof Key for Code Exchange (PKCE)

- PKCE (RFC 7636) is an extension to the Authorization Code flow to prevent CSRF and authorization code injection attacks.

- PKCE is not a replacement for a client secret, and PKCE is recommended even if a client is using a client secret.

SECRET

- In the context of an authorization code grant, the main purpose of the client secret is to prevent client impersonation.
- It prevents a malicious client from pretending to be a legitimate client in order to get an access token from the resource owner under false pretenses.

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Scopes

- Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account.
- An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.

Bearer Token

- Bearer Tokens are the predominant type of access token used with OAuth 2.0.
- A Bearer Token is an opaque string, not intended to have any meaning to clients using it.
- Some servers will issue tokens that are a short string of hexadecimal characters, while others may use structured tokens such as JSON Web Tokens.

Front Channel ++ back channel

- Front Channel
    - The front channel flow is used by the client application to obtain an authorization code grant.
    - Less Secure Cahnnel.
- Back Channel
    - The back channel is used by the client application to exchange the authorization code grant for an access token (and optionally a refresh token).
    - Highly Secure Channel

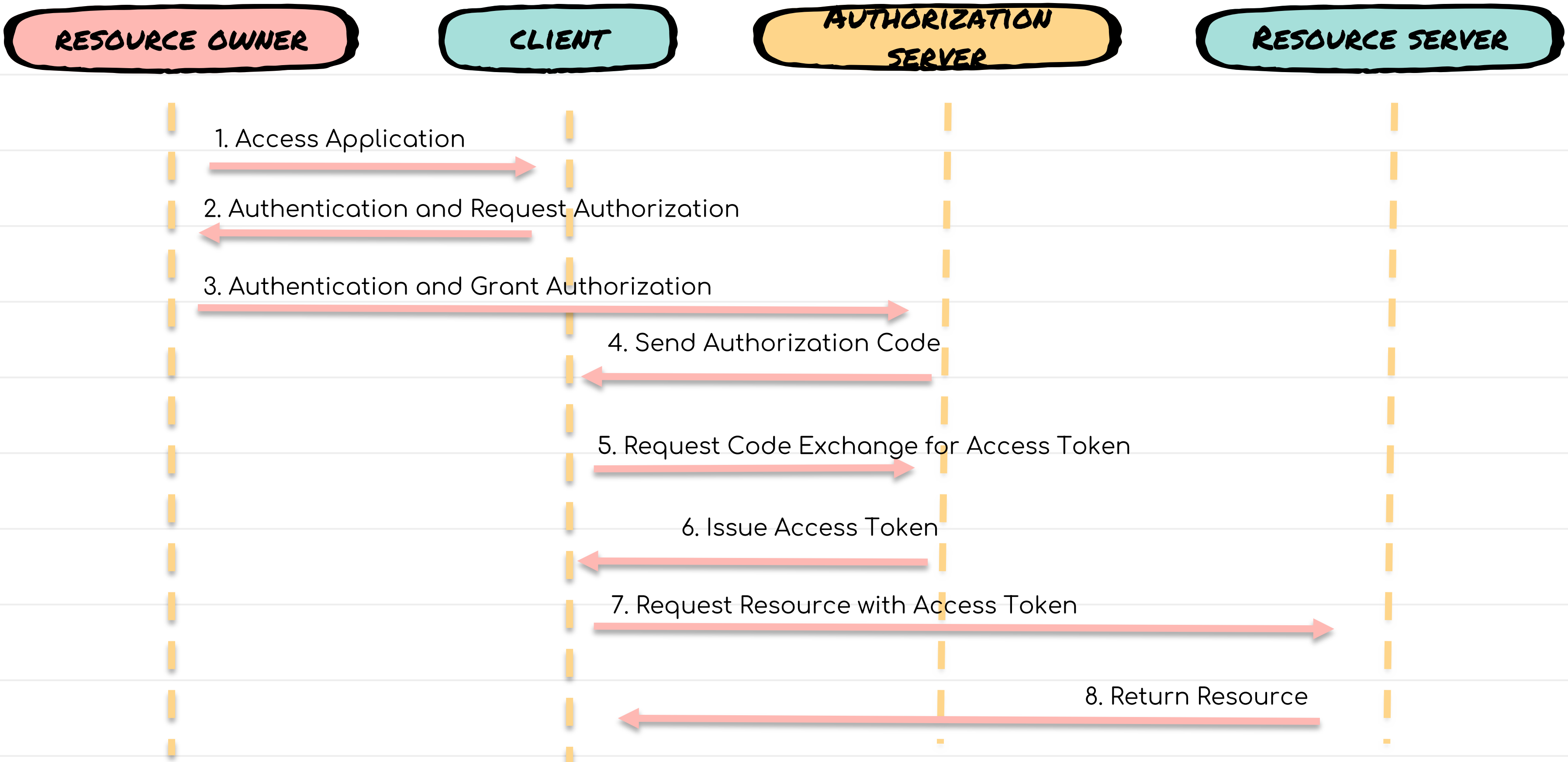
Application Registration

- You must register your application with the service.
- This is done through a registration form in the developer or API portion of the service's website, where you will provide the following information (and probably details about your application):
  - Application Name
  - Application Website
  - Redirect URI or Callback URL

Oauth 2.0 flows

- Authorization Code (Front Channel + Back Channel)
- Implicit (front channel only)
  - Strongly discouraged as it is extremely challenging to implement the Implicit flow securely.
- Authorization Code Flow with PKCE (Proof Key for Code Exchange)

AUTHORIZATION CODE FLOW



RESOURCE OWNER

CLIENT

AUTHORIZATION SERVER

RESOURCE SERVER

1. Access Application
2. Authentication and Request Authorization
3. Authentication and Grant Authorization
4. Send Authorization Code
5. Request Code Exchange for Access Token
6. Issue Access Token
7. Request Resource with Access Token
8. Return Resource

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Authorization flow with PKCE

- The Authorization Code flow with PKCE is similar to the standard Authorization Code flow with an extra step at the beginning and an extra verification at the end.

- Client generates a code verifier followed by a code challenge.

- Client sends this code, along with the code verifier to Authorization Server.

- Authorization Server evaluates the PKCE code.

Authorization flow with PKCE

- Code Verifier
  - Random URL-safe string with a minimum length of 43 characters.
- Code Challenge
  - Base64 URL-encoded SHA-256 hash of the code verifier.

Authorization flow with PKCE

- Client saves the `code_verifier` for later and sends the `code_challenge` along with the authorization request to the Authorization Server.
- The authorization server will hash the provided plaintext and confirm that the hashed version corresponds with the hashed string that was sent in the initial authorization request.

Authorization Request Parameters

- The following parameters are used to make the authorization request.

- A query string with the below parameters needs to be build, appending that to the application's authorization endpoint.

Authorization Request Parameters

- `response_type=code`
    - response_type is set to code indicating that you want an authorization code as the response.

- `client_id`
    - The client_id is the identifier for your app.
    - You will have received a client_id when first registering your app with the service.

Authorization Request Parameters

- `redirect_uri` (optional)
    - The redirect_uri may be optional depending on the API, but is highly recommended.
    - This is the URL to which you want the user to be redirected after the authorization is complete.
    - This must match the redirect URL that you have previously registered with the service.

Authorization Request Parameters

- `scope` (optional)
  - Include one or more scope values (space-separated) to request additional levels of access.
  - The values will depend on the particular service.

Exchange the authorization code for an access token

- To exchange the authorization code for an access token, the app makes a POST request to the service's token endpoint.
- The request will have the following parameters.
    - `grant_type` (required)
        - The grant_type parameter must be set to "authorization_code".
    - `code` (required)
        - This parameter is for the authorization code received from the authorization server which will be in the query string parameter "code" in this request.

Verifying Access Tokens

- The resource server will be getting requests from applications with an HTTP Authorization header containing an access token.

- The resource server needs to be able to verify the access token to determine whether to process the request, and find the associated user account, etc.

Verifying Access Tokens

- If you’re using self-encoded access tokens, then verifying the tokens can be done entirely in the resource server without interacting with a database or external servers.

- If your tokens are stored in a database, then verifying the token is simply a database lookup on the token table.

OpenID Connect

- The OAuth 2.0 framework explicitly does not provide any information about the user that has authorized an application.

- OAuth 2.0 is a delegation framework, allowing third-party applications to act on behalf of a user, without the application needing to know the identity of the user.

OpenID Connect

- OpenID Connect takes the OAuth 2.0 framework and adds an identity layer on top.
- It provides information about the user, as well as enables clients to establish login sessions.

Oauth ++ OpenID Connect

- OAUTH
  - Granting access to the API
  - Getting access to user data in other systems.
  - Authorization
- OpenID Connect
  - Logging the user in
  - Authentication

KEYCLOAK

- Keycloak is an open-source Identity and Access Management solution targeted towards modern applications and services.
    - Single-Sign-On (SSO)
    - Identity Brokering and Social Login
    - User Federation

KEYCLOAK

- Client Adapters
- Admin Console
- Account Management Console

REALMS

- A realm is a space where you manage objects, including users, applications, roles, and groups.
    - A user belongs to and logs into a realm.
- One Keycloak deployment can define, store, and manage as many realms as there is space for in the database.

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Main points

- Security underlies an entire enterprise. It provides a shield that makes the application invulnerable.

- Transcendental Consciousness is characterized by the quality of invincibility, which means one cannot be overcome or overpowered.