Mastering the Spring Framework

Chapter 5: Securing REST Endpoints with Spring Security

Chapter Objectives

In this chapter, we will:

- Introduce Spring Security
- Secure REST endpoints using Spring Security
- Provide overview of OAuth 2.0

Chapter Concepts

Spring Boot Security

OAuth 2.0

Spring Boot Security

- Spring Security provides a rich set of security features
  - HTTP BASIC authentication headers
  - HTTP Digest authentication headers
  - HTTP X.509 client certificate exchange
  - LDAP
  - Form-based authentication
  - OpenID authentication
  - Much more …
    - https://projects.spring.io/spring-security/
- If Spring Security is found on the class path, then HTTP Basic authentication will be applied to all endpoints

Setting Up Spring Security

- Spring Security must be added to the build file

```
…
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
…
```

POJO with properties matching JSON property names

Configuring Spring Security

- Configuring Spring Security for authentication requires two steps:
  1. Defining an authentication manager
     - Defines where username, password, and role information is stored
       - In memory
       - JDBC
       - LDAP
  2. Configure which URLs are restricted
- Configuration of both steps is possible in Java

Step 1: Defining an Authentication Manager

- Define a method that configures `AuthenticationManagerBuilder`
- For memory manager define username, password, and role(s)

Provides configuration for Spring Security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
                                                        throws Exception {
        auth
            .inMemoryAuthentication()
            .withUser("john").password("smith").roles("USER")
                .and()
            .withUser("admin").password("admin").roles("USER", "ADMIN");
    }
}
```

Authentication Manager

- When an authentication manager is configured, the following process occurs:
  1. On receipt of a username and password, Spring Security verifies that they are valid
  2. If valid, then the list of roles for that user is obtained and a security context created
  3. Security context is used to determine if user has permission – correct role – to access endpoint
     - If no roles configured, a valid username and password combination are enough to access the endpoint

Step 2: Configure URL Access

- Spring's `HttpSecurity` class provides a DSL for configuring URL access
  - Allows URL patterns defined using Ant style pattern matchers

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
           .and()
        .authorizeRequests()
           .antMatchers("/audit/**").hasRole("ADMIN")
           .antMatchers("/testobjects/**").hasAnyRole("USER", "ADMIN")
           .antMatchers("/echo").authenticated()
           .anyRequest().permitAll()
           .and()
        .httpBasic()
           .and()
        .csrf().disable()
        .headers().frameOptions().disable();
}
```

Don't create session

Access for specific role

Any authenticated user

Non-authenticated access

Chapter Concepts

Spring Boot Security

OAuth 2.0

OAuth 2.0

- Open authorization protocol, which enables applications to access each other's data
  - Data is referred to as a resource
- Grants an application access to protected resources only for specific uses; e.g., read-only
  - Can be time-limited access
- Useful with REST services to allow one service to securely access another service
- OAuth 2.0 defines the following roles:
  - Resource owner (user or application)
  - Resource server
  - Client application
  - Authorization server

OAuth 2.0 Definitions

- OAuth2 flows are using **tokens** as the means to pass authorization information around
- Types of tokens:
  - Access token
    - Used to make API requests to Resource Server on behalf of a user
    - Represents authorization of client application to access specific parts of a user's data
  - Refresh token
    - Used to obtain new access token in lieu of expired one
  - Authorization code
    - Intermediate token issued by Authorization Server and exchanged for access token
- Client application registration information:
  - Client id
    - 'Unique' public identifier for client application
  - Client secret
    - Private secret known only to the application and the authorization server

OAuth 2.0 Roles

- Most common responsibilities for each of the OAuth 2.0 roles are:
  - Authorization Server:
    - Maintains client application registration information
    - Issues access and refresh tokens to Client Application
  - Client Application
    - Retrieves access/refresh tokens from Authorization Server
    - Makes requests to 'Resource Server' on behalf of the 'Resource Owner' (User)
    - Coordinates authentication flow between Authorization Server, Resource Owner, and Resource Server
  - Resource Server
    - API server to access user resources (e.g., Google Drive, Google Docs, etc.)
    - Validates access tokens with help of Authorization Server
      - Unless access tokens are 'self-encoded'
  - Resource Owner
    - Person who is giving access to some portion of their account

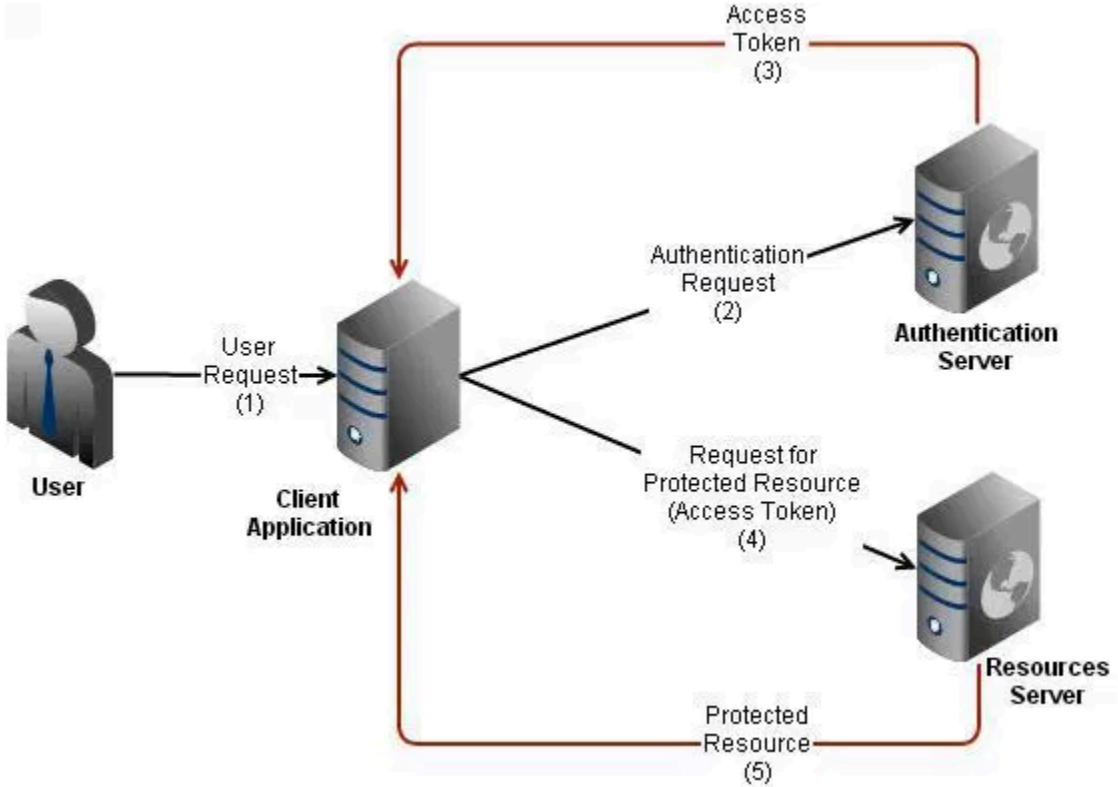
OAuth2 Grants

- OAuth2 specification describes five grants (or 'methods') for a client application to acquire an access token ('grant_type'):
  - Authorization code grant
    - Most common flow (login via Facebook/Google)
    - Doubles up as a single sign-on mechanism
  - Implicit grant
    - Similar to the authorization code grant, but without refresh token or client secret
  - Resource owner credentials grant
    - Client passes through to Authorization Server user credentials (usually a username and password)
  - Client credentials grant
    - Server to server authentication; specific user's permission to access data is not required, only client ID and secret are used
  - Refresh token grant
    - Requests new access token to replace expired one

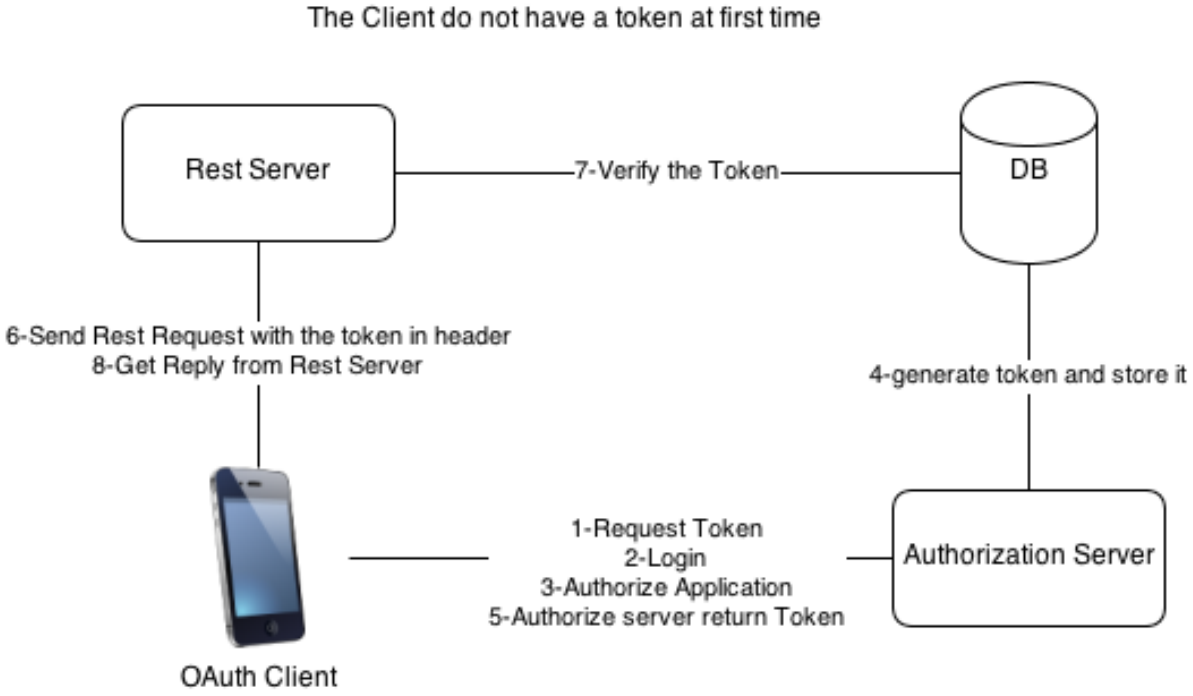
'Generic' OAuth 2.0 Flow

- Accessing a resource using OAuth 2.0 proceeds as follows:
  1. Client Application is registered with Authorization Server
     - Client ID and Client Secret are provided to Client Application 'out of band'
  2. User or application makes request to a Client Application
     - Could be a REST service
  3. Client Application makes request to Authorization Server, passing over:
     - Client ID, Client Secret (+ user credentials for some flows), access scope
  4. Authorization Server responds with access token (+ refresh token for some flows)
     - As part of user authentication, some flows might respond with an intermediate 'authorization code' which client will exchange for access token via separate call
  5. Client Application sends request to Resource Server together with access token
  6. Resource Server validates access token and returns requested resource
     - Validation can be done either against Authorization Server or by validating token signature for self-encoded tokens (normally via JWT/JWS)
  7. Client Application updates access token from Authorization Server before access token expires (for some flows)

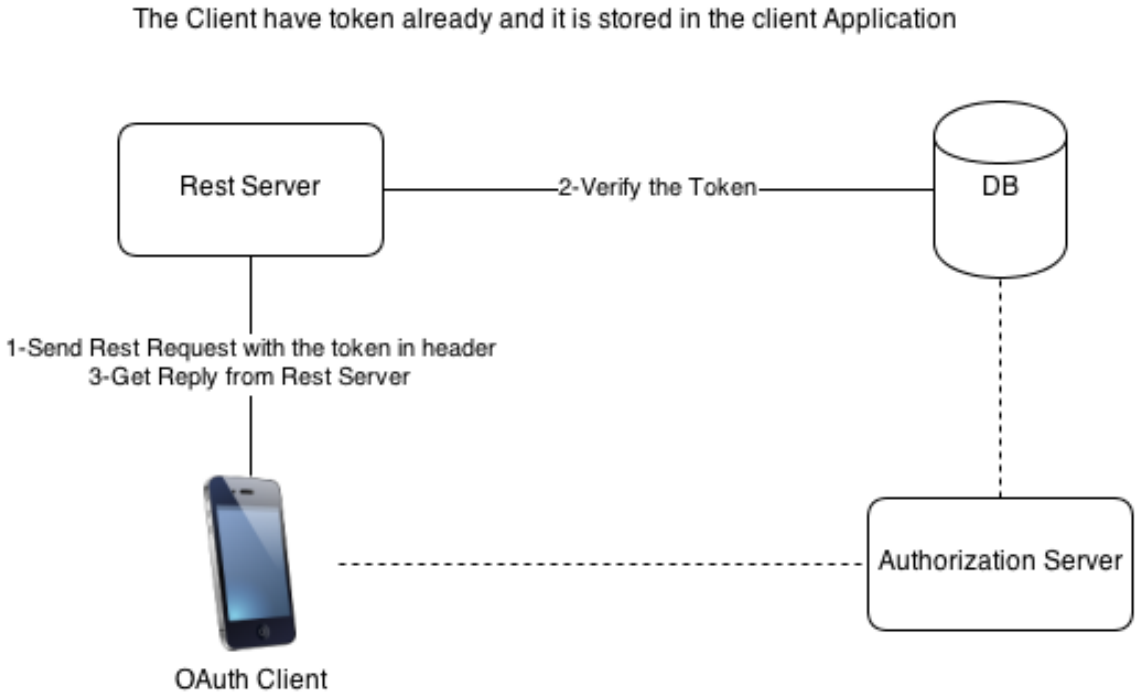
‘Generic’ OAuth 2.0 Flow (continued)



Step 1: Getting Token



Step 2: Getting Protected Resource with Token



Spring and OAuth 2.0

- Spring security provides support for OAuth 2.0
- Example provided at https://spring.io/guides/tutorials/spring-boot-oauth2/
- Simple way to explore OAuth 2.0 authentication – Google OAuth 2.0 Playground: https://developers.google.com/oauthplayground/

Chapter Summary

In this chapter, we have:

- Introduced Spring Security
- Secured REST endpoints using Spring Security
- Provided overview of OAuth 2.0