**Spring Security ACL Demo**

This project demonstrates how to use Spring Security's Access Control List (ACL) functionality to implement fine-grained authorization for domain objects.

Overview

Spring Security ACL allows you to control access to individual objects in your application, rather than just at the application level (e.g., by roles). This is useful when you need to define very specific permissions, such as "User A can edit Document 1, but not Document 2."

This demo provides a basic setup to help you understand and experiment with Spring Security ACL.

**Prerequisites**

* Java Development Kit (JDK): Version 21 or higher.
* Maven: Version 3.6.0 or higher.
* Database: A running instance of a database (e.g., MySQL, PostgreSQL, H2). The **application is configured to use H2 by default**.
* Basic understanding of Spring Security: Familiarity with concepts like authentication, authorization, roles, and Spring Security annotations.

**Getting Started**

1. Download this zip file in your <my-app> folder. Unzip and import it into your favourite IDE.
2. Configure the database (Optional):
   * If you want to use a database other than the default H2, update the database configuration in src/main/resources/application.properties. For example, for MySQL:
   * spring.datasource.url=jdbc:mysql://localhost:3306/your\_database\_name
   * spring.datasource.username=your\_mysql\_username
   * spring.datasource.password=your\_mysql\_password
   * spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
   * spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
   * Make sure the database exists and the user has the necessary privileges.
3. Build the project:
4. mvn clean install
5. Run the application: I used the following:  
   "**<** Maven install directory path**>**" **clean install** -Dmaven.test.skip=true

The application will start, and you can access it at   
1. <http://localhost:9192/my_login.html>  
2. Enter user name adminUser/adminPass1 as username /password.   
3. Enter some document name , and document content in the respective text boxes, Hit “Create”.  
You should be able to see some success message.  
4. If you **used another**, already created, user(without create permissions) credentials userReader1/userPass1, you **won’t be allowed** to create the document!  
These users are created in the file **DatabseInitilaizer.java**

**Additional stuff, not necessarily related to this application, but to Spring Security ACL**

**Key Features and How to Test Them**

This demo includes functionality for managing documents with ACLs. Here's how you can test the key features:

1. Authentication:
   * The application uses Spring Security for authentication. You'll need to authenticate before accessing secured resources. See the src/main/resources/data.sql file for default users and roles. Here are a couple of users:
     + userReader1 / password (Has the ROLE\_USER role)
     + adminUser1 / password (Has the ROLE\_ADMIN role)
   * Use a tool like curl or Postman to make HTTP requests. Include the Authorization header with a Basic Auth token. For example, in curl:
   * curl -v -u userReader1:password http://localhost:8080/some/secured/endpoint
2. Document Creation:
   * The POST /cDoc endpoint creates a new document. Only **users with** the ROLE\_ADMIN role can access this endpoint.
   * To test:
   * curl -v -u adminUser1:password \
   * -X POST http://localhost:8080/cDoc \
   * -d "pName=MyDocument&pContent=This is the content"
     + Verify that the response has a 201 Created status.
     + Try the same request with userReader1 and confirm that it works.
     + Try with a user that does not have either role and confirm that you get a 403 Forbidden error.
3. Securing the Document Creation Endpoint:
   * The @PreAuthorize("hasRole('ROLE\_ADMIN') or hasRole('ROLE\_USER')") annotation on the createDocument method in the controller enforces this authorization. This ensures that only users with the specified roles can create documents. This is the core of the ACL demo.

**Important Files and Classes**

* src/main/java/com/example/springsecurityacl/config/SecurityConfig.java: Spring Security configuration, including ACL setup.
* src/main/java/com/example/springsecurityacl/service/DocumentAclService.java: Service class that handles ACL operations for documents. This is where the ACLs are created and managed.
* src/main/java/com/example/springsecurityacl/web/controller/DocumentController.java: Controller for handling document-related requests.
* src/main/resources/application.properties: Application properties, including database configuration.
* src/main/resources/data.sql: Initial data, including users, roles, and initial ACL setup. This file is crucial for setting up your initial testing environment.

**Understanding the ACL Structure**

Spring Security ACL uses these core concepts:

* Object Identity (OI): Uniquely identifies the domain object (e.g., a specific document).
* Security Identity (SID): Represents who is granted permissions (a user or a role).
* Permission: Defines what actions can be performed on the object (e.g., read, write, delete).
* Access Control Entry (ACE): связывает OI, SID и Permission, specifying who can do what to which object.
* Acl: Holds all the ACEs for a given OI.

**Further Exploration**

* Custom Permissions: Explore how to define custom permissions beyond the basic ones (read, write, etc.).
* ACL Inheritance: Learn how ACLs can inherit permissions from parent objects.
* AclService: Dive deeper into the AclService interface and its implementations for managing ACLs.
* Method Security: Investigate how to use Spring Security ACL with method security annotations (@PreAuthorize, @PostAuthorize, @PreFilter, @PostFilter) for fine-grained control.

Enjoy experimenting with Spring Security ACL!