Securing REST Application with Spring Security Lab

Purpose

In this lab you will gain experience with using Spring Security to secure the REST application through authentication and authorization.

Learning Outcomes

What you will learn:

- 1. How to implement authentication
- 2. How to restrict access to URLs and/or HTTP methods
- 3. How to test secured application

Specific subjects you will gain experience with:

- 1. Spring Security and the SecurityFilterChain
- 2. Built-in and custom UserDetailsService
- 3. Security test annotations such as @WithMockUser and @WithUserDetails

You will be using the 42-security-rest project.

Estimated time to complete: 45 minutes.

Prerequisites

The prerequisites are included as part of the Introduction to the Spring Professional Learning Path course Lab Setup lesson.

If you already completed it, you should be ready to do this lab. If not, assuming you already have JDK 11 or 17 installed and Java IDE, you will need to do following:

- 1. Download the lab codebase zip file.
- 2. Once you have downloaded the file, unzip it under a directory of your choice. The unzipped directory core-spring-labfiles/lab contains the lab projects.
- 3. Run ./mvnw clean install (if you plan on using Maven as your build tool).

Use Case

• Implement authentication and authorization against REST application

Quick Instructions

If you are already knowledgeable with the lesson concepts, you may consider jumping right to the code, and execute the lab in form of embedded TODO comments. Instructions on how to view them are at the *Using TODO Tasks* article.

If you aren't sure, try the TODO instructions first and refer to the lab instructions by TODO number if you need more help.

Instructions

The RestWsApplication you built in the previous lab is not secured. You are going to secure it through authentication and authorization.

Explore Default Behaviour of Spring Boot Security

TODO-01: Verify the presence of Spring security dependencies

When Spring Boot sees spring-boot-starter-security on the classpath, it will set up default authentication and authorization - it will set up a single up and all endpoints are secured.

The spring-security-test dependency provides testing support.

TODO-02a: Observe the default security behaviour of the Spring Boot application using a browser

Spring Boot application relies on Spring Security's content-negotiation strategy to determine whether to use Basic authentication or Form-based authentic

When you are using a browser, Spring Boot application will use Form-based authentication.

- Using Chrome Incognito browser, access http://localhost:8080/accounts and observe that a login page gets displayed
- Enter user in the Username field and Spring Boot generated password into the Password field and verify that the accounts get displayed (If the brow keeps displaying the login page, make sure to use Chrome Incognito browser.)

```
INFO: o.s.b.web.servlet.FilterRegistrationBean - Mapping filter: 'httpTraceFilter' to: [/*]
INFO: o.s.b.web.servlet.FilterRegistrationBean - Mapping filter: 'webMvcMetricsFilter' to: [/*]
INFO: o.s.b.w.servlet.ServletRegistrationBean - Servlet dispatcherServlet mapped to [/]
INFO: o.s.b.a.w.s.WelcomePageHandlerMapping - Adding welcome page: class path resource [static/index.html]
INFO: o.s.b.a.s.s.UserDetailsServiceAutoConfiguration -

Using generated security password: f49a49f1-df8a-4da8-b3e8-89fb204bda24

INFO: o.s.s.web.DefaultSecurityFilterChain - Creating filter chain: org.springframework.security.web.util.matcher.A
INFO: o.s.b.d.a.OptionalLiveReloadServer - LiveReload server is running on port 35729
```

Figure 1: Password output to stdout

• Access http://localhost:8080/logout and click Log out button

TODO-02b: Observe the default security behaviour of the Spring Boot application using curl command

When you are sending a REST request using curl command, Spring Boot application will use Basic authentication.

When you are sending a REST request without valid username/password pair, the application will respond with 401 Unauthorized.

- Open a terminal window
- Run "curl -i localhost:8080/accounts" and observe 401 Unauthorized response
- Run "curl -i -u user:<Spring-Boot-Generated-Password> localhost:8080/accounts" and observe a successful response

Enable Web Security

Spring Boot's default setup is deliberately minimal. It is not intended for production. Instead, you should configure you own security rules.

TODO-03: Import security configuration class

The RestSecurityConfig class, which is located under config package, needs to be explicitly imported since it will not be found through component-sc from the accounts package.

Configure Authentication and Authorization

Since Web security is enabled, we are ready to configure authentication and authorization.

TODO-04: Configure authorization

The most common form of authorization (access control) is through the usage of roles.

You are going define 3 roles - USER, ADMIN, and SUPERADMIN with the following access control rules:

- Allow DELETE on the /accounts resource (or any sub-resource) for SUPERADMIN role only
- Allow POST or PUT on the /accounts resource (or any sub-resource) for ADMIN or SUPERADMIN role only
- Allow GET on the /accounts resource (or any sub-resource) for all roles USER, ADMIN, SUPERADMIN
- Allow GET on the /authorities resource for all roles USER, ADMIN, SUPERADMIN

TODO-05: Configure authentication

Here you are going to set up in-memory UserDetailsService and define three users with a corresponding set of roles assigned.

- user/user with USER role
- admin/admin with USER and ADMIN roles
- superadmin/superadmin with USER, ADMIN, and SUPERADMIN

TODO-06a: Perform security testing against MVC layer

Now You are going to test if the security configuration works against MVC layer using @WebMvcTest and @WithMockUser annotations.

Open the AccountControllerTests test class. The tests cover the following scenarios:

- Using an invalid user credential to perform any operation should result in 401 Unauthorized response
- Using USER role, you can only perform read operation
- Using ADMIN role, you can perform create/update operation
- Using SUPERADMIN role, you can perform delete operation

Most tests are already provided. Make sure you understand what each test is meant to verify.

Remove the @Disabled annotation from each test and run them. They should all pass.

TODO-06b: Write a test that verifies that a user with USER role is not permitted to perform a POST operation.

This is an opportunity for you to write a simple security test code.

TODO-07a: Perform security testing against a running server

Now You are going to test if the security configuration works using end-to-end testing against the application running over the embedded server.

Open the AccountClientTests test class. The tests cover the similar set of scenarios mentioned above.

Remove the @Disabled annotation from each test and run them. They should all pass.

TODO-07b: Write a test that verifies that user/user is not permitted to create a new Account

Retrieve Authorities (Roles) for the Logged-In User

Once authentication is successful, security context is being maintained in the ThreadLocal during the processing of a request.

This allows any method in the call stack in the same thread of execution can access the security context even if the security context is not explicitly passe around as an argument to those methods.

Using a ThreadLocal in this way is quite safe if care is taken to clear the thread after the present principal's request is processed. Of course, Spring Sec takes care of this for you automatically so there is no need to worry about it.

You are going to access the security context via SecurityContextHolder class.

In the AccountService class, use the SecurityContextHolder to get Security context, which in turn can be used to get Authentication object, which used to get authorities.

An example code might look like the following:

Implement Method Security

TODO-09: Add method security annotation to a method

Here you are going to use method security so that the method will be invoked only when both of the following conditions are met:

- The logged-in user belongs to ADMIN role
- The value of the username request parameter of the request URL matches the value of the principal's username. This condition ca specified using SpEL (Spring Expression language).

An example could be either

```
@PreAuthorize("hasRole('ADMIN') && #username == principal.username")
```

or

```
@PreAuthorize("hasRole('ADMIN') && #username == authentication.name")
```

TODO-10: Enable method security

Method security needs to be enabled via the <code>@EnableMethodSecurity</code> annotation. The <code>prePostEnabled</code> attribute is set to <code>true</code> by default, this allows the usage of the <code>@PreAuthorize</code> annotation.

An example code is as follows:

@EnableMethodSecurity

TODO-11: Test the method security using a browser or curl

Now let's see if the method security works against a running application.

- Re-run this application
- Use user/user credential to access the method, and verify the application returns 403 Forbidden response since user/user does belong to neithe ADMIN nor SUPERADMIN role.

```
curl -i -u user:user http://localhost:8080/authorities?username=user
```

• Use admin/admin and verify that the roles are displayed successfully

```
curl -i -u admin:admin http://localhost:8080/authorities?username=admin
```

• Use superadmin/superadmin and verify that the roles are displayed successfully

curl -i -u superadmin:superadmin http://localhost:8080/authorities?username=superadmin

TODO-12a: Perform method security testing

The AccountServiceMethodSecurityTest class contains tests that you are going to use for performing method security testing.

Remove the @Disabled annotation from each test and run them. They should all pass.

TODO-12b: Write a test that verifies that getting authorities using http://localhost:8080/authorities?username=superadmin with superadmin/superadmin credential should return three roles ROLE_SUPERADMIN, ROLE_ADMIN, and ROLE_USER.

This is an opportunity for you to write a test code.

Create Custom UserDetailsService (Optional)

Do the remaining steps only if you have extra time.

TODO-13: Create custom UserDetailsService

Open the CustomUserDetailsService class. Note that it needs to implement loadUserByUsername method of the UserDetailsService interface.

The custom UserDetailsService maintains UserDetails of two users:

- mary/mary with USER role and
- joe/joe with USER and ADMIN role

Typically you would use some kind of persistence storage for maintaining user data but in this lab, for the sake of simplicity, you are going to just return he coded UserDetails object given a username.

TODO-14: Add authentication based upon the custom UserDetailsService

Annotate the CustomUserDetailsService class with @Component to make it a Spring managed bean. Spring Boot will automatically discover it and it wi used as the UserDetailsService.

TODO-15: Verify that the newly added custom UserDetailsService works

Since the custom UserDetailsService maintains UserDetails on two users, mary/mary and joe/joe, you can now verify that you can access a secure resource using the identity of one of these two users.

curl -i -u mary:mary http://localhost:8080/accounts/0

curl -i -u joe:joe http://localhost:8080/accounts/0

TODO-16: Perform security testing for the two users added through custom UserDetailsService

Create Custom AuthenticationProvider (Optional)

TODO-17: Create custom AuthenticationProvider

Note that it needs to implement ${\tt AuthenticationProvider}$ interface.

Typically you will use a custom authentication system for the verification of the passed identity.

In this lab, we are just faking it so that it handles a user with the following identity.

• spring/spring with ROLE_ADMIN role

TODO-18: Add authentication based upon the custom AuthenticationProvider

Now you are going to add authentication based upon the custom AuthenticationProvider using AuthenticationManagerBuilder object.

auth.authenticationProvider(new CustomAuthenticationProvider());

So now there are 6 users configured - user/user, admin/admin, superadmin/superadmin, mary/mary, joe/joe, and spring/spring.

TODO-19: Verify that the newly added custom AuthenticationProvider works

Since the custom AuthenticationProvider can validate the identity of spring/spring, you can now verify that you can access a secured resource usi

curl -i -u spring:spring http://localhost:8080/accounts/0

TODO-20: Perform security testing for the user added through custom AuthenticationProvider

Summary

In this lab, we secured Spring REST application with authentication and auhtorization.

We also secured a service-layer method using method security.

Finally we configured custom UserDetailsService and AuthenticationProvider.

Congratulations, you are done with the lab!

Summary Instructors

In this lesson you saw Spring Security in action.

Give Feedback

Help us improve by sharing your thoughts.

Give Feedback



Guides | Courses | Learning Path | Community | Instructors | About | Give Feedback

Contact | FAQs | Terms | Privacy | Your California Privacy Rights

Unlock your full potential with Spring courses designed by experts.