Hello Spring Security Java Config

Rob Winch – Version 3.2.5.RELEASE

This guide provides instructions on how to add Spring Security to an existing application without the use of XML.

Table of Contents

Setting up the sample

This section outlines how to setup a workspace within Spring Tool Suite (STS) so that you can follow along with this guide. The next section outlines generic steps for how to apply Spring Security to your existing application. While you could simply apply the steps to your existing application, we encourage you to follow along with this guide as is to reduce the complexity.

Obtaining the sample projects

Extract the Spring Security Distribution to a known location and remember it as *SPRING_SECURITY_HOME*.

Import the insecure sample application

In order to follow along, we encourage you to import the insecure sample application into your IDE. You may use any IDE you prefer, but the instructions in the guide will assume you are using Spring Tool Suite (STS).

The completed sample application can be found at *SPRING_SECURITY_HOME*/samples/helloworld-jc

- If you do not have STS installed, download STS from https://spring.io/tools
- Start STS and import the sample applications into STS using the following steps:
  - **File→Import**
  - **Existing Maven Projects**
  - Click **Next >**
  - Click **Browse…**
  - Navigate to the samples (i.e. *SPRING_SECURITY_HOME*/samples/insecure) and click **OK**
  - Click **Finish**

Running the insecure application

In the following exercise we will be modifying the *spring-security-samples-insecure* application. Before we make any changes, it is best to verify that the sample works properly. Perform the following steps to ensure that *spring-security-samples-insecure* works.

- Right click on the *spring-security-samples-insecure* application
- Select **Run As→Run on Server**
- Select the latest tc Server
- Click **Finish**

Verify the application is working by ensuring a page stating **TODO Secure this** is displayed at http://localhost:8080/sample/

Once you have verified the application runs, stop the application server using the following steps:

- In the Servers view select the latest tc Server
- Click the stop button (a red square) to stop the application server

---

Securing the application

Before securing your application, it is important to ensure that the existing application works as we did in Running the insecure application. Now that the application runs without security, we are ready to add security to our application. This section demonstrates the minimal steps to add Spring Security to our application.

Updating your dependencies

Spring Security GA releases are included within Maven Central, so no additional Maven repositories are necessary.

In order to use Spring Security you must add the necessary dependencies. For the sample we will add the following Spring Security dependencies:

pom.xml

```
<dependencies>
  <!-- ... other dependency elements ... -->
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>3.2.5.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>3.2.5.RELEASE</version>
  </dependency>
</dependencies>
```

After you have completed this, you need to ensure that STS knows about the updated dependencies by:

- Right click on the *spring-security-samples-insecure* application
- Select **Maven→Update project…**
- Ensure the project is selected, and click **OK**

Creating your Spring Security configuration

The next step is to create a Spring Security configuration.

- Right click the *spring-security-samples-insecure* project the Package Explorer view
- Select **New→Class**
- Enter *org.springframework.security.samples.config* for the **Package**
- Enter *SecurityConfig* for the **Name**

- Click **Finish**
- Replace the file with the following contents:

src/main/java/org/springframework/security/samples/config/SecurityConfig.java

```
package org.springframework.security.samples.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.*;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

The name of the configureGlobal method is not important. However, it is important to only configure AuthenticationManagerBuilder in a class annotated with either `@EnableWebSecurity`, `@EnableWebMvcSecurity`, `@EnableGlobalMethodSecurity`, or `@EnableGlobalAuthentication`. Doing otherwise has unpredictable results.

The SecurityConfig will:

- Require authentication to every URL in your application
- Generate a login form for you
- Allow the user with the **Username** *user* and the **Password** *password* to authenticate with form based authentication
- Allow the user to logout
- CSRF attack prevention
- Session Fixation protection
- Security Header integration
    - HTTP Strict Transport Security for secure requests
    - X-Content-Type-Options integration
    - Cache Control (can be overridden later by your application to allow caching of your static resources)
    - X-XSS-Protection integration
    - X-Frame-Options integration to help prevent Clickjacking
- Integrate with the following Servlet API methods
    - HttpServletRequest#getRemoteUser()
    - HttpServletRequest.html#getUserPrincipal()
    - HttpServletRequest.html#isUserInRole(java.lang.String)
    - HttpServletRequest.html#login(java.lang.String, java.lang.String)
    - HttpServletRequest.html#logout()

Registering Spring Security with the war

We have created the Spring Security configuration, but we still need to register it with the war. This can be done using the following steps:

- Navigate to the **Package Explorer** view
- Right click the **org.springframework.security.samples.config** package within the **spring-security-samples-insecure** project
- Select **New→Class**
- Enter *SecurityWebApplicationInitializer* for the **Name**
- Click **Finish**
- Replace the file with the following contents:

src/main/java/org/springframework/security/samples/config/SecurityWebApplicationInitializer.java

```
package org.springframework.security.samples.config;

import org.springframework.security.web.context.*;

public class SecurityWebApplicationInitializer
      extends AbstractSecurityWebApplicationInitializer {

    public SecurityWebApplicationInitializer() {
        super(SecurityConfig.class);
    }
}
```

The `SecurityWebApplicationInitializer` will do the following things:

- Automatically register the springSecurityFilterChain Filter for every URL in your application
- Add a ContextLoaderListener that loads the SecurityConfig.

Since we were not already using Spring, this is a simple way to add our SecurityConfig. If we were already using Spring, then we should add our SecurityConfig with the reset of our Spring configuration (i.e. a subclass of AbstractContextLoaderInitializer or AbstractDispatcherServletInitializer) and use the default constructor instead.

Exploring the secured application

Start the server as we did in Running the insecure application Now when you visit http://localhost:8080/sample/ you will be prompted with a login page that is automatically generated by Spring Security.

Authenticating to the secured application

Try entering an invalid username and password:

- **Username** *invalid*
- **Password** *invalid*

You should see an error message stating that authentication failed. Now try entering a valid username and password:

- **Username** *user*
- **Password** *password*

You should now see the page that we wanted to secure.

The reason we can successfully authenticate with **Username** *user* and **Password** *password* is because that is what we configured in our SecurityConfig.

Displaying the user name

Now that we have authenticated, let's update the application to display the username. Update the body of index.jsp to be the following:

src/main/webapp/index.jsp

```
<body>
  <div class="container">
    <h1>This is secured!</h1>
    <p>
      Hello <b><c:out value="${pageContext.request.remoteUser}"/></b>
    </p>
  </div>
</body>
```

The `<c:out />` tag ensures the username is escaped to avoid XSS vulnerabilities Regardless of how an application renders user inputed values, it should ensure that the values are properly escaped.

Refresh the page at http://localhost:8080/sample/ and you will see the user name displayed. This works because Spring Security integrates with the Servlet API methods

Logging out

Now that we can view the user name, let's update the application to allow logging out. Update the body of index.jsp to contain a log out form as shown below:

src/main/webapp/index.jsp

```
<body>
  <div class="container">
    <h1>This is secured!</h1>
    <p>
      Hello <b><c:out value="${pageContext.request.remoteUser}"/></b>
    </p>
    <c:url var="logoutUrl" value="/logout"/>
    <form class="form-inline" action="${logoutUrl}" method="post">
      <input type="submit" value="Log out" />
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
    </form>
  </div>
</body>
```

In order to help protect against CSRF attacks, by default, Spring Security Java Configuration log out requires:

- the HTTP method must be a POST
- the CSRF token must be added to the request You can access it on the ServletRequest using the attribute _csrf as illustrated above.

If you were using Spring MVC's tag library or Thymeleaf, the CSRF token is automatically added as a hidden input for you.

Refresh the page at http://localhost:8080/sample/ and you will see the log out button. Click the logout button and see that the application logs you out successfully.

Conclusion

You should now know how to secure your application using Spring Security without using any XML. To learn more refer to the Spring Security Guides index page.