Securing a Web Application in Spring Boot

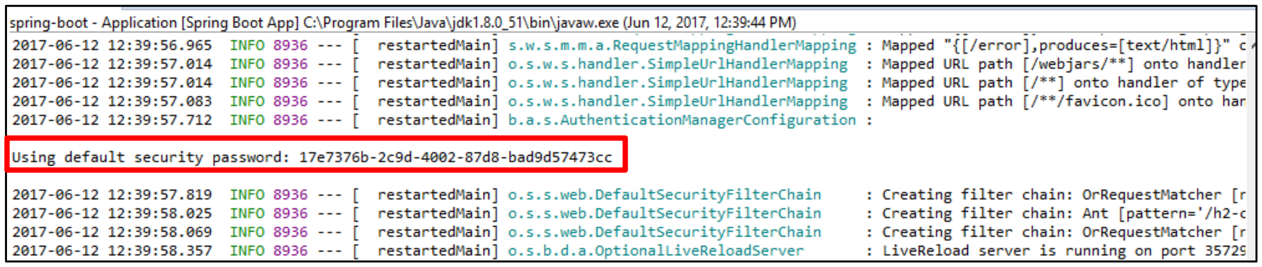
This guide walks you through the process of securing a Web application in Spring boot using Spring Security.

1. Open existing Spring boot Web application project.
2. Open the pom.xml file and add the following starter pom for security.

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

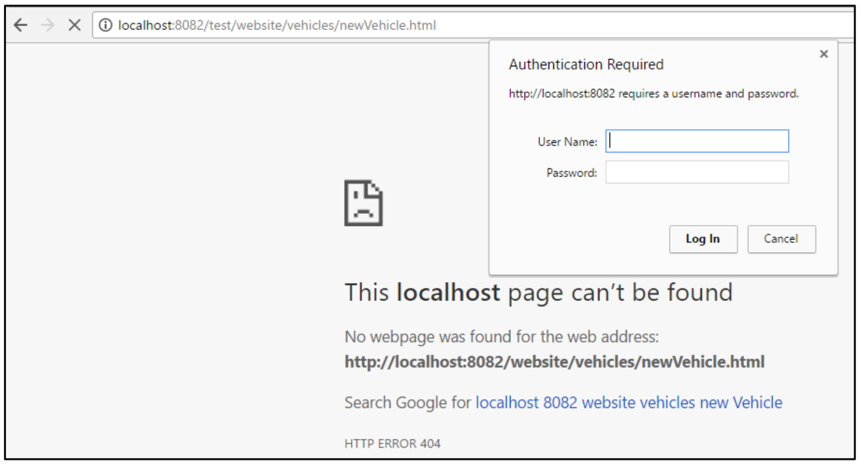
If Spring Security is on the classpath then Web applications will be secure by default with '**basic**' authentication on all HTTP endpoints.

3. Build the application and in the log you will notice a default password and configuration of **DefaultSecurityFilterChain** with some common URL patterns.



```
spring-boot - Application [Spring Boot App] C:\Program Files\Java\jdk1.8.0_51\bin\javaw.exe (Jun 12, 2017, 12:39:44 PM)
2017-06-12 12:39:56.965  INFO 8936 --- [  restartedMain] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" c
2017-06-12 12:39:57.014  INFO 8936 --- [  restartedMain] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler
2017-06-12 12:39:57.014  INFO 8936 --- [  restartedMain] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type
2017-06-12 12:39:57.083  INFO 8936 --- [  restartedMain] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto har
2017-06-12 12:39:57.712  INFO 8936 --- [  restartedMain] b.a.s.AuthenticationManagerConfiguration :

Using default security password: 17e7376b-2c9d-4002-87d8-bad9d57473cc

2017-06-12 12:39:57.819  INFO 8936 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: OrRequestMatcher [r
2017-06-12 12:39:58.025  INFO 8936 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: Ant [pattern='/h2-c
2017-06-12 12:39:58.069  INFO 8936 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: OrRequestMatcher [r
2017-06-12 12:39:58.357  INFO 8936 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
```

4. Now access the URL for **newVehicle.html** page:



localhost:8082/test/website/vehicles/newVehicle.html

Authentication Required

http://localhost:8082 requires a username and password.

User Name:

Password:

This **localhost** page can't be found

No webpage was found for the web address: **http://localhost:8082/website/vehicles/newVehicle.html**

Search Google for localhost 8082 website vehicles new Vehicle

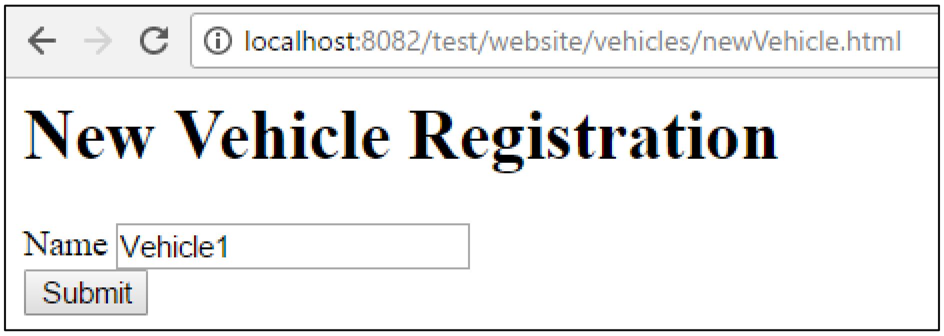
HTTP ERROR 404

Figure 1: Basic Authentication

Default user credentials:

- User Name: **user**
- Password: **17e7376b-2c9d-4002-87d8-bad9d57473cc** (copy from the console log)

Now, the page will be accessed.as shown in the figure.



localhost:8082/test/website/vehicles/newVehicle.html

New Vehicle Registration

Name Vehicle1

Figure 2: Access to the URL

Thus, all the pages are protected with the basic security. Hence, the default configuration is very useful.

However, we can customize to provide our own configuration to set up the component, such as form. To we can override default configuration provided by Spring boot.

Configure a Spring Security using Java Config.

1. Create a new package named **com.training.springboot.configuration** in the project.
2. Create a class named **SecurityConfiguration.java** under the package.
3. Annotate the class with necessary annotations
4. Extend the class with **WebSecurityConfigurerAdapter**.

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter{

}
```

The **SecurityConfiguration** class is annotated with **@EnableWebSecurity** to enable Spring Security's web security support and provide the Spring MVC integration.

It also extends **WebSecurityConfigurerAdapter** and overrides a couple of its methods to set some specifics of the web security configuration. This will help us to plug in our custom configurations.

For authentication, we need to override a method called **configure()** which takes an instance of `AuthenticationManagerBuilder`.

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {

		auth.inMemoryAuthentication()
		.withUser("user")
		.password("secret")
		.roles("USER", "ADMIN");
	}
}
```

The **configure (AuthenticationManagerBuilder)** method, it sets up an in-memory user store with a single user. That user is given a username of "**user**", a password of "**secret**", and a role of "**USER**" and "**ADMIN**".

For authorization of a particular resource, we need to override the **configure()** method with **HttpSecurity** as a parameter.

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

	// Authentication
	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {

		. . .
	}

	// Authorization
	@Override
	protected void configure(HttpSecurity http) throws Exception {
		// authorization rules

      http.antMatcher("/**").authorizeRequests()
          .anyRequest()
          .hasRole("USER")

         .and()


     .formLogin()
     .loginPage("/login.jsp")
     .failureUrl("/login.jsp?error=1")
     .loginProcessingUrl("/login")
     .permitAll()
     .and()
     .logout()
```

```
        .logoutSuccessUrl("/website/vehicles/list.html");
        }

}
```

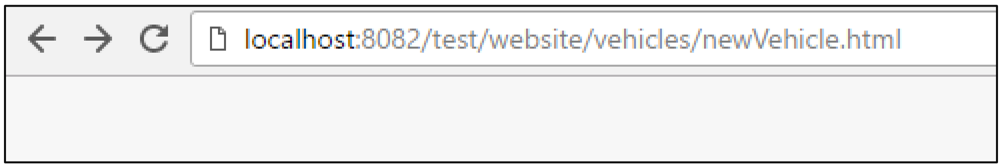
The **configure(HttpSecurity)** method defines which URL paths should be secured and which should not. Specifically, the "/**" paths are configured to authorize each resource in the application.

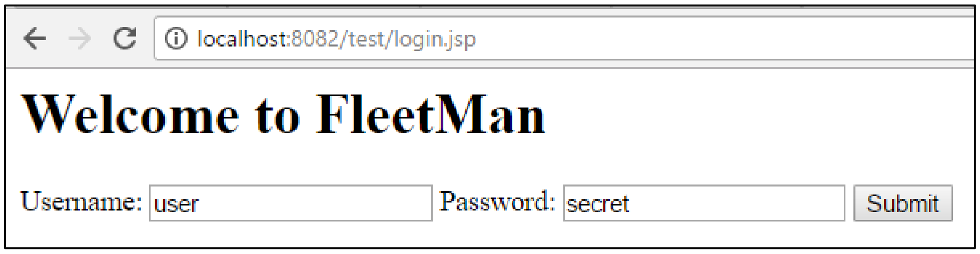
When a user successfully logs in, they will be redirected to the previously requested page that required authentication. There is a custom "/login" page specified by **loginPage()**, and everyone is allowed to view it. The error page is appended with the error code.

5. Add the **login.jsp** page in the under **webapp** folder.

```
 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>
        <head>
                <title>Login to FleetMan</title>
        </head>

        <body>

                <h1><strong>Welcome to FleetMan</strong></h1>

        <c:url value="/login" var="loginUrl"/>

                <c:if test="${param.error != null}">
                        <p>Bad username/password</p>
                </c:if>

                <form:form action="${loginUrl}" method="post">
                 <label>Username:</label>
                <input type="text" name="username" value="rac"/>
                 <label>Password:</label>
                <input type="text" name="password" value="secret"/>
               <input type="submit"/>
                </form:form>
        </body>
</html>
```

6. Build the Spring boot application and access the URL, **newVehicle.html**.


localhost:8082/test/website/vehicles/newVehicle.html



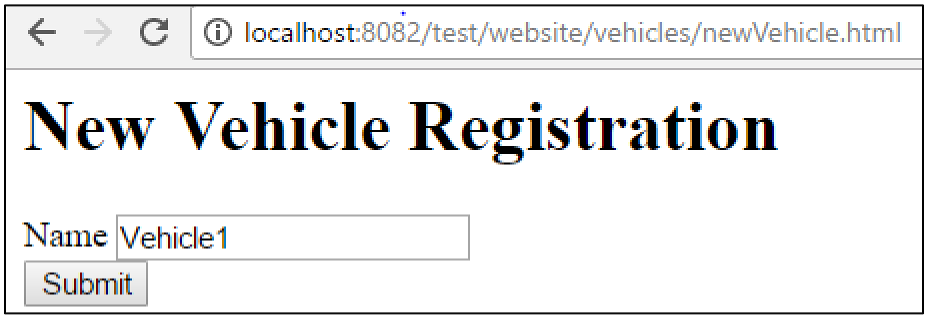
localhost:8082/test/login.jsp

Welcome to FleetMan

Username: user Password: secret Submit

Figure: Customized Login Page

7. Clicking on **Submit** button takes you to the **newVehicle.html**.



localhost:8082/test/website/vehicles/newVehicle.html

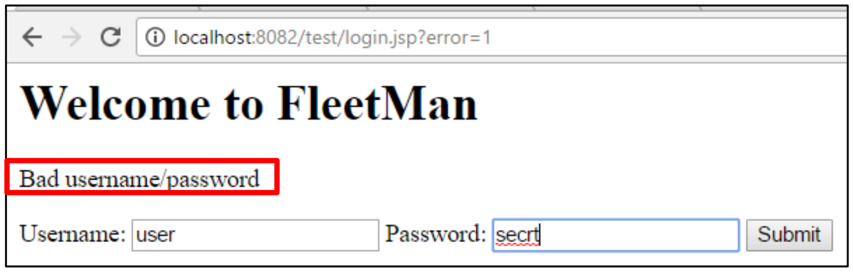
New Vehicle Registration

Name Vehicle1

Submit

Figure: Correct Page

8. In case, if the user name and password were not correct, then the **login.jsp** with error is displayed.



localhost:8082/test/login.jsp?error=1

Welcome to FleetMan

Bad username/password

Username: user Password: secrt Submit

Figure: Error Page