Create a minimally secured web app

Project Information:

* **Group**: io.pivotal.workshop
* **ArtifactId**: simple-security
* **Dependencies**: web, security

1. Create the project using the [Spring Initializr](http://start.spring.io/) or IntelliJ (***File → New → Spring Starter Project***).
2. Open the project in IntelliJ (or any other IDE).
3. Notice the **Spring Boot Starters** added to the ***build.gradle*** file:
   * ***spring-boot-starter-security***
   * ***spring-boot-starter-web***
4. compile "org.springframework.boot:spring-boot-starter-security"
5. compile "org.springframework.boot:spring-boot-starter-web"
6. testCompile "org.springframework.boot:spring-boot-starter-test"
7. Create a new web controller class (***io.pivotal.workshop.controller.MainController.java***) with the following requirements:
   * A REST controller GET endpoint at "/"
   * This endpoint returns the string "Hello World! - Secured"
8. **package** io.pivotal.workshop.controller;
9. **import** org.springframework.web.bind.annotation.GetMapping;
10. **import** org.springframework.web.bind.annotation.RestController;
11. **@RestController**
12. **public** **class** **MainController** {
13. **@GetMapping**("/")
14. **public** String **helloSecurity**() {
15. **return** "Hello World! - Secured";
16. }
17. }
18. You can use the terminal and execute: ./gradlew bootRun
19. Take a look at the output and look for the ***password***. You should see a print out: *Using default security password: ...*. Copy the ***password*** to the clipboard because you will use it!
20. Open a Browser window and go to [http://localhost:8080](http://localhost:8080/), it will ask you for *username* and *password*. The *username* is: **user** and the *password* is the one you have in the clipboard.
21. Once you enter the username/password you will be able to see the web page.
22. In application.properties / application.yml change the configured password to the super secure password: password.
23. security.user.password=password
24. Restart the application and open a browser window to [http://localhost:8080](http://localhost:8080/), it will ask you for *username* and *password*. The *username* is: **user** and the *password* is now **password**.

We currently have an application that is secured with one user named user, but what if we wanted more users.

1. Create a new SecurityConfig class: ***io.pivotal.workshop.config.SecurityConfig.java***:
2. **package** io.pivotal.workshop.config;
3. **import** org.springframework.beans.factory.annotation.Autowired;
4. **import** org.springframework.context.annotation.Configuration;
5. **import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
6. **import** org.springframework.security.config.annotation.web.builders.HttpSecurity;
7. **import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
8. **import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
9. **@EnableWebSecurity**
10. **@Configuration**
11. *// Configuring this class prevents the Spring Boot autoconfiguration*
12. *// from happening.*
13. **public** **class** **SecurityConfig** **extends** **WebSecurityConfigurerAdapter** {
14. **@Autowired**
15. **protected** **void** **configureUser**(AuthenticationManagerBuilder auth) **throws** Exception {
16. auth
17. .inMemoryAuthentication()
18. .withUser("billy").password("bob").roles("USER")
19. .and()
20. .withUser("admin").password("password").roles("ADMIN");
21. }
22. *// We do not want the default behavior of form authentication*
23. *// before HTTP Basic authentication we get*
24. *// from WebSecurityConfigurerAdapter.*
25. **@Override**
26. **protected** **void** **configure**(HttpSecurity http) **throws** Exception {
27. http.authorizeRequests()
28. .anyRequest().fullyAuthenticated()
29. .and()
30. .httpBasic();
31. }
32. }
33. Remove security.user.password=password from application.properties/application.yml.
34. Start the application in a terminal window: ./gradlew bootRun.
35. With curl do a request using the first user, then with the second user
36. curl localhost:8080 -u admin:password
37. curl localhost:8080 -u billy:bob
38. You created a minimal secured web application with HTTP Basic authentication.

Secure the actuator endpoints

With the spring-boot-actuator lab finished (you will change it).

1. In your build.gradle, for the spring-boot-actuator project, add the spring-boot-starter-security dependency.
2. compile "org.springframework.boot:spring-boot-starter-security"
3. Use the terminal and execute: ./gradlew bootRun
4. Use curl to request the /mappings with default credentials
5. curl localhost:8080/mappings -u <username>:<password>

By default sensitive endpoints are secured when spring-boot-starter-security is on your class path.

1. Stop your application.
2. In your application.properties/application.yml add endpoints.mappings.sensitive=false.
3. Use the terminal and execute: ./gradlew bootRun
4. Use curl to request the /mappings without credentials
5. curl localhost:8080/mappings

You should now see a list of all the mappings even though we have not authenticated with our application.

1. Navigate to <http://localhost:8080/>. Notice we still have basic auth. Next let's allow our users to see our greeting message without authenticating.
2. Stop your application.
3. In your application.properties/application.yml remove or comment out endpoints.mappings.sensitive=false, then add security.basic.enabled=false.
4. Restart the application and use curl to request the following urls without credentials:
5. curl localhost:8080
6. curl localhost:8080/mappings

Even though we have disabled HTTP basic auth, actuator endpoints are still secured.

You have seen how actuator defaults to having secured endpoints and disabled security on an endpoint allowing public access.

Challenge

Use a different username and password for the actuator endpoints.

Create an application secured with OAuth2

For some background on OAuth2, see this article from [Digital Ocean](https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2).

First let's create a [resource server](http://projects.spring.io/spring-security-oauth/docs/oauth2.html#resource-server-configuration). This will have a simple endpoint that will return the string Hello - Secured when authenticated.

Building our resource server

Project Information:

* **Group**: io.pivotal.workshop
* **ArtifactId**: oauth-resource-server
* **Dependencies**: web, security

1. Create the project using the [Spring Initializr](http://start.spring.io/) or IntelliJ (***File → New → Spring Starter Project***).
2. Open the project in IntelliJ (or any other IDE).
3. Add the spring-security-oauth2 dependency to your build.gradle.
4. compile "org.springframework.security.oauth:spring-security-oauth2"
5. compile "org.springframework.boot:spring-boot-starter-security"
6. compile "org.springframework.boot:spring-boot-starter-web"
7. Configure the application as a resource server.
8. **package** io.pivotal.workshop;
9. **import** org.springframework.boot.SpringApplication;
10. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
11. **import** org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;
12. **@EnableResourceServer**
13. **@SpringBootApplication**
14. **public** **class** **OAuthResourceServerApplication** {
15. **public** **static** **void** **main**(String[] args) {
16. SpringApplication.run(OAuthResourceServerApplication.class, args);
17. }
18. }
19. Create a new web controller class (***io.pivotal.workshop.controller.ResourceController.java***) with the following specifications:
    * A REST controller with one GET endpoint at "/".
    * This endpoint returns the string "Hello - Secured".
20. **package** io.pivotal.workshop.controller;
21. **import** org.springframework.web.bind.annotation.GetMapping;
22. **import** org.springframework.web.bind.annotation.RestController;
23. **@RestController**
24. **public** **class** **ResourceController** {
25. **@GetMapping**("/")
26. **public** String **helloSecured**() {
27. **return** "Hello - Secured";
28. }
29. }
30. Set server port to 8081 in your application.properties/application.yml file.
31. Start up the application: ./gradlew bootRun
32. We are going to use curl as our client to access our resource server.
33. In your terminal execute: curl localhost:8081
    * What do you see?
    * What does this mean?

Authorization Server

1. Download the Authorization server from [here](https://cognizant-pal-v6.apps.cumuluslabs.io/spring-boot-developer/security/auth-server.jar). The source code can also be found in the class source repository.
2. Start up this application: java -jar auth-server.jar
3. Take a look at the output, you will see the **security.oauth2.client.clientId** and then **security.oauth2.client.secret**properties listed with their respective ids. You need to save them because you will use them.
4. Open a new Terminal and execute the following command:
5. curl localhost:8080/oauth/token -d scope=**read** -d grant\_type=password -d username=user -d password=password -u CLIENTID:SECRET

Replace the **CLIENTID** and **SECRET** with the values you got from the output.

**For Example**:

curl localhost:8080/oauth/token -d scope=**read** -d grant\_type=password -d username=user -d password=password -u 77562a14-647b-493f-aea7-ae96d6b8535a:132cda75-aea8-478c-b9dd-c05675b02730

1. From the above command you will receive a JSON response with an ***access\_token*** value. You will use that for the next call to the resource server.

Authenticating with the resource server

1. In the resource-server application add to the application.properties/application.yml file:
   * **security.oauth2.client.client-id=CLIENTID FROM AUTHORIZATION SERVER**
   * **security.oauth2.client.client-secret=SECRET FROM AUTHORIZATION SERVER**
   * **security.oauth2.resource.token-info-uri=http://localhost:8080/oauth/check\_token**

This will allow your resource server to act as a client of the resource server and ask the authorization server questions about the validity of the token.

1. Restart the server to pick up on the above property changes, then use curl as our client and gain access by executing the following command:
2. curl localhost:8081 -H "Authorization: Bearer ACCESS\_TOKEN"

**For example:**

curl localhost:8081 -H "Authorization: Bearer 5291486c-0d69-4c24-94da-30c20fc14024"

You should see the "Hello - Secured" response.

You created a resource server secured with OAuth2.