Security

Requirements

Lab Requirements

Purpose of this lab

- Get familiar with Spring Boot Security on a simple web app.
- Get familiar with Spring Boot Security for Actuator
- Get familiar with Spring Boot Security OAuth2 for securing an API.
- Estimated time: 40 minutes

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 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***

```
compile "org.springframework.boot:spring-boot-start
er-security"
compile "org.springframework.boot:spring-boot-start
er-web"
testCompile "org.springframework.boot:spring-boot-s
tarter-test"
```

4. Create a new web controller class (***io.pivotal.workshop.controller.MainController.java***) with the following requirements:

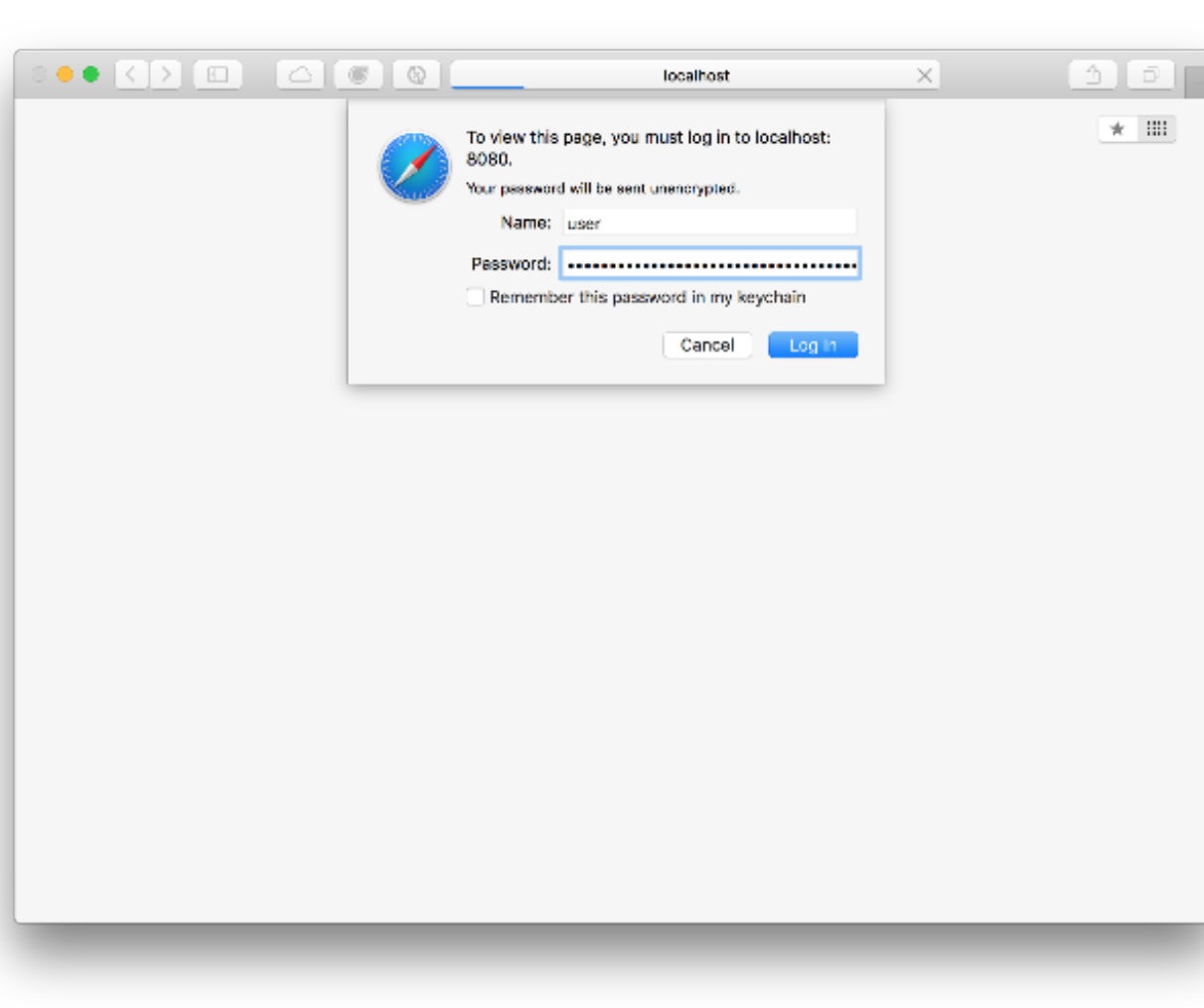
   - A REST controller GET endpoint at "/"
   - This endpoint returns the string "Hello World! - Secured"

```
package io.pivotal.workshop.controller;

import org.springframework.web.bind.annotation.GetM
apping;
import org.springframework.web.bind.annotation.Rest
Controller;

@RestController
public class MainController {

    @GetMapping("/")
    public String helloSecurity() {
        return "Hello World! - Secured";
    }
}
```

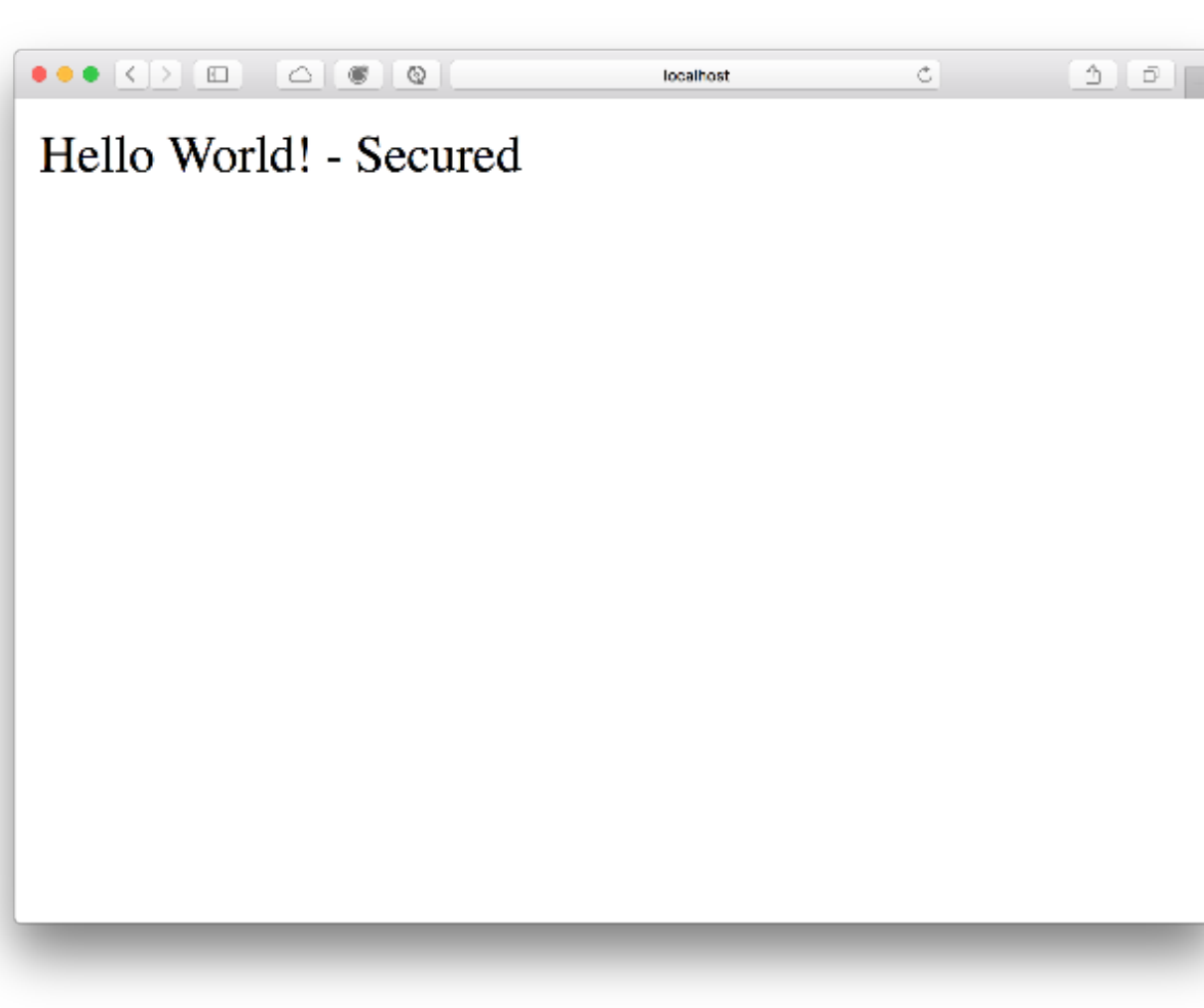
5. You can use the terminal and execute: `./gradlew bootRun`

6. 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!

7. Open a Browser window and go to 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.



8. Once you enter the username/password you will be able to see the web page.



9. In application.properties / application.yml change the configured password to the super secure password: `password`.

```
security.user.password=password
```

10. Restart the application and open a browser window to 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.

11. Create a new `SecurityConfig` class: ***io.pivotal.workshop.config.SecurityConfig.java***:

```
package io.pivotal.workshop.config;

import org.springframework.beans.factory.annotation
.Autowired;
import org.springframework.context.annotation.Confi
guration;
import org.springframework.security.config.annotati
on.authentication.builders.AuthenticationManagerBui
lder;
import org.springframework.security.config.annotati
on.web.builders.HttpSecurity;
import org.springframework.security.config.annotati
on.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotati
on.web.configuration.WebSecurityConfigurerAdapter;

@EnableWebSecurity
@Configuration
// Configuring this class prevents the Spring Boot
autoconfiguration
// from happening.
public class SecurityConfig extends WebSecurityConf
igurerAdapter {

    @Autowired
    protected void configureUser(AuthenticationMana
gerBuilder auth) throws Exception {
        auth
                .inMemoryAuthentication()
                .withUser("billy").password("bob").
roles("USER")
                .and()
                .withUser("admin").password("passwo
rd").roles("ADMIN");
    }

    // We do not want the default behavior of form
authentication
    // before HTTP Basic authentication we get
    // from WebSecurityConfigurerAdapter.
    @Override
    protected void configure(HttpSecurity http) thr
ows Exception {
        http.authorizeRequests()
                .anyRequest().fullyAuthenticated()
                .and()
                .httpBasic();
    }
}
```

12. Remove `security.user.password=password` from application.properties/application.yml.

13. Start the application in a terminal window: `./gradlew bootRun`.

14. With curl do a request using the first user, then with the second user

```
curl localhost:8080 -u admin:password
curl localhost:8080 -u billy:bob
```

15. 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.

```
compile "org.springframework.boot:spring-boot-start
er-security"
```

2. Use the terminal and execute: `./gradlew bootRun`

3. Use curl to request the `/mappings` with default credentials

```
curl localhost:8080/mappings -u <username>:<passwor
d>
```

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

```
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:

```
curl localhost:8080
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.

First let's create a resource server. 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 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.

```
compile "org.springframework.security.oauth:spring-
security-oauth2"
compile "org.springframework.boot:spring-boot-start
er-security"
compile "org.springframework.boot:spring-boot-start
er-web"
```

4. Configure the application as a resource server.

```
package io.pivotal.workshop;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.Sprin
gBootApplication;
import org.springframework.security.oauth2.config.a
nnotation.web.configuration.EnableResourceServer;

@EnableResourceServer
@SpringBootApplication
public class OAuthResourceServerApplication {

    public static void main(String[] args) {

        SpringApplication.run(OAuthResourceServerAp
plication.class, args);
    }
}
```

5. 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".

```
package io.pivotal.workshop.controller;

import org.springframework.web.bind.annotation.GetM
apping;
import org.springframework.web.bind.annotation.Rest
Controller;


@RestController
public class ResourceController {

    @GetMapping("/")
    public String helloSecured() {

        return "Hello - Secured";
    }

}
```

6. Set server port to `8081` in your application.properties/application.yml file.

7. Start up the application: `./gradlew bootRun`

8. We are going to use `curl` as our client to access our resource server.

9. In your terminal execute: `curl localhost:8081`

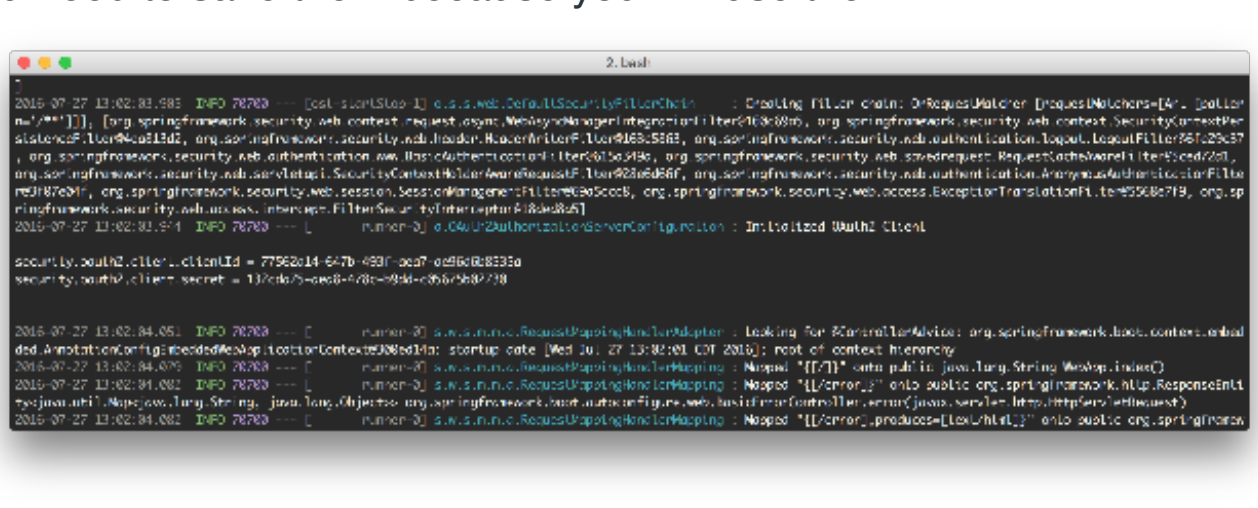
   - What do you see?
   - What does this mean?

Authorization Server

1. Download the Authorization server from here. 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:

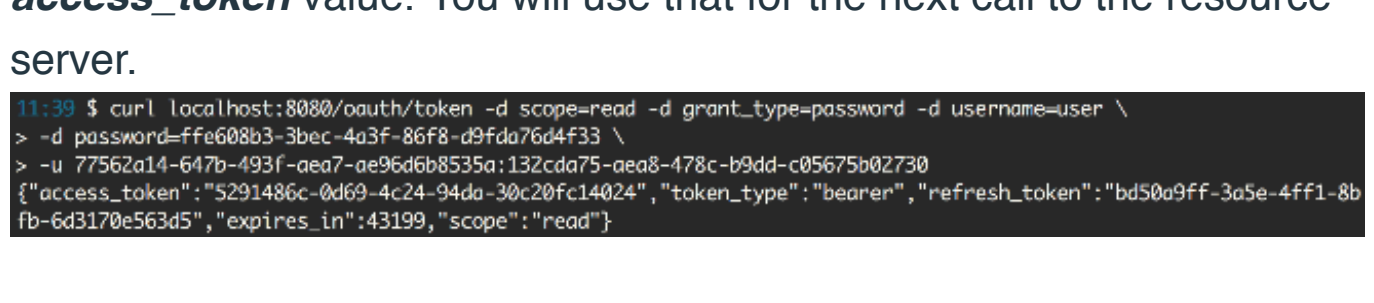
```
curl localhost:8080/oauth/token -d scope=read -d
grant_type=password -d username=user -d password=pa
ssword -u CLIENTID:SECRET
```

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=pa
ssword -u 77562a14-647b-493f-aea7-ae96d6b8535a:132c
da75-aea8-478c-b9dd-c05675b02730
```

5. 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.tokenInfoUri=http://localhost:8080/oauth/c**

   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.

2. Let's use curl as our `client` and gain access by executing the following command:

```
curl localhost:8081 -H "Authorization: Bearer ACCES
S_TOKEN"
```

For example:

```
curl localhost:8081 -H "Authorization: Bearer 52914
86c-0d69-4c24-94da-30c20fc14024"
```

You should see the "Hello World! - Secured" response.

> You created a resource server secured with OAuth2.