FREE eBook

LEARNING spring-security

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-security

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: spring-security

It is an unofficial and free spring-security ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official spring-security.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with spring-security

Remarks

This section provides an overview of what spring-security is, and why a developer might want to use it.

It should also mention any large subjects within spring-security, and link out to the related topics. Since the Documentation for spring-security is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
4.2.2	2017-03-02
3.2.10	2016-12-22
4.2.1	2016-12-21
4.1.4	2016-12-21
4.2.0	2016-11-10

Examples

Installation or Setup

Detailed instructions on getting spring-security set up or installed.

Spring Securitiy to protect REST API endpoints

Add below entries in `pom.xml`.

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>3.1.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>3.1.0.RELEASE</version>
```

```
</dependency>
```

Important for Spring version greater than 3.1:

Bean creation error for `org.springframework.security.filterChains` comes when you are using Spring version higher than 3.1 and have not added dependencies manually for `spring-aop`, `spring-jdbc`, `spring-tx` and `spring-expressions` in your `pom.xml`.

Add below entries in Spring context. We want to protect two REST endpoints (helloworld & goodbye). Adjust XSD version according to Spring version.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:security="http://www.springframework.org/schema/security"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
                       http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.1.xsd
                       http://www.springframework.org/schema/security
                       http://www.springframework.org/schema/security/spring-security-
3.1.xsd">

    <security:http auto-config='true' create-session="never">
        <security:intercept-url pattern="/helloworld/**" access="ROLE_USER" />
        <security:intercept-url pattern="/goodbye/**" access="ROLE_ADMIN" />
        <security:intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <security:http-basic />
    </security:http>

    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="username1" password="password1"
                    authorities="ROLE_USER" />
                <security:user name="username2" password="password2"
                    authorities="ROLE_ADMIN" />
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>
</beans>
```

Add below entries in `web.xml`.

```
<!-- Spring security-->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

```
</listener>

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:security-context.xml</param-value>
</context-param>
```

Spring-Security using spring-boot and JDBC Authentication

Suppose you want to prevent unauthorized users to access the page then you have to put barrier to them by authorizing access. We can do this by using spring-security which provides basic authentication by securing all HTTP end points. For that you need to add spring-security dependency to your project, in maven we can add the dependency as:

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

Here's a security configuration that ensures that only authenticated users can access.

```
@Configuration
@Order(SecurityProperties.ACCESS_OVERRIDE_ORDER)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
DataSource datasource;

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest()
            .fullyAuthenticated()
            .and()
        .formLogin()
            .loginPage("/login")
            .failureUrl("/login?error")
            .permitAll()
            .and()
        .logout()
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login?logout")
            .permitAll()
            .and()
        .csrf();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication().dataSource(datasource).passwordEncoder(passwordEncoder());
}

@Bean
public PasswordEncoder passwordEncoder() {
    PasswordEncoder encoder = new BCryptPasswordEncoder();
    return encoder;
```

```
}
}
```

Configuration	Description
`@Configuration`	Indicates that the class can be used by the Spring IoC container as a source of bean definitions.
`@Order (SecurityProperties.ACCESS_OVERRIDE_ORDER)`	Override the access rules without changing any other autoconfigured features. Lower values have higher priority.
`WebSecurityConfigurerAdapter`	The `SecurityConfig` class extends and overrides a couple of its methods to set some specifics of the security configuration.
`@Autowired` of `DataSource`	Provide factory for connections to the physical data source.
`configure(HttpSecurity)`	Overridden method defines which URL paths should be secured and which should not.
`.authorizeRequests().anyRequest() .fullyAuthenticated()`	Indicates to spring that all request to our application requires to be authenticated.
`.formLogin()`	Configures a form based login
`.loginPage("/login").failureUrl("/login?error").permitAll()`	Specifies the location of the log in page and all users should be permitted to access the page.
`.logout().logoutUrl("/logout")` `.logoutSuccessUrl("/login?logout").permitAll()`	The URL to redirect to after logout has occurred. The default is /login?logout.
`.csrf()`	Used to prevent Cross Site Request Forgery, CSRF

Configuration	Description
	protection is enabled (default).
`configure(AuthenticationManagerBuilder){}`	Overridden method to define how the users are authenticated.
`.jdbcAuthentication().dataSource(datasource)`	Indicates to spring that we are using JDBC authentication
`.passwordEncoder(passwordEncoder())`	Indicates to spring that we are using a password encoder to encode our passwords. (A bean is created to return the choice of password Encoder, we are using BCrypt in this case)

Notice that we have not configured any table name to be used or any query, this is because spring security by default looks for the below tables:

```
create table users (
  username varchar(50) not null primary key,
  password varchar(255) not null,
  enabled boolean not null) ;

create table authorities (
  username varchar(50) not null,
  authority varchar(50) not null,
  foreign key (username) references users (username),
  unique index authorities_idx_1 (username, authority));
```

Insert the following rows into the above tables:

```
INSERT INTO authorities(username,authority)
VALUES ('user', 'ROLE_ADMIN');

INSERT INTO users(username,password,enabled)
VALUES('user', '$2a$10$JvqOtJaDys0yoXPX9w47YOqu9wZr/PkN1dJqjG9HHAzMyu9EV1R4m', '1');
```

The **username** in our case is `user` and the **password** is also `user` encrypted with BCrypt algorithm

Finally, Configure a datasource in the application.properties for spring boot to use:

```
spring.datasource.url = jdbc:mysql://localhost:3306/spring
spring.datasource.username = root
spring.datasource.password = Welcome123
```

Note: Create and configure a login controller and map it to the path `/login` and point your login page to this controller

Hello Spring Security

> **Note 1:** You need some prior knowledge about ***java servlet page(JSP)*** and ***Apache Maven*** before you start this examples.

Start the web server (like ***Apache tomcat***) with existing web project or create one.

Visit the ***index.jsp***.

Anybody can access that page, it's insecure!

Securing application

1. Update Maven dependencies

Adding dependencies to your pom.xml file

pom.xml

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>4.0.1.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>4.0.1.RELEASE</version>
</dependency>
```

> **Note 1:** If you're not using "Spring" in your project before, there's no dependency about "spring-context". This example will use xml config with "spring-context". So add this dependency too.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>4.2.2.RELEASE</version>
</dependency>
```

> **Note 2:** If you're not using JSTL in your project before, there's no dependency about that. This example will use JSTL in jsp page. So add this dependency too.

```
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>javax.servlet.jsp.jstl</artifactId>
```

```
  <version>1.2.1</version>
</dependency>
```

---

2. Make Spring Security Configuration File

Make folder name "spring" inside the "WEB-INF" folder and make security.xml file. Copy and paste from next codes.

WEB-INF/spring/security.xml

```
<b:beans xmlns="http://www.springframework.org/schema/security"
         xmlns:b="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                             http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd">

  <http />

  <user-service>
    <user name="stackoverflow" password="pwd" authorities="ROLE_USER" />
  </user-service>

</b:beans>
```

---

3. Update web.xml

Update your web.xml inside the "WEB-INF" folder

WEB-INF/web.xml

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

> **Note:** If you're not using "Spring" in your project before, there's no configurations about Spring contexts load. So add this parameter and listener too.

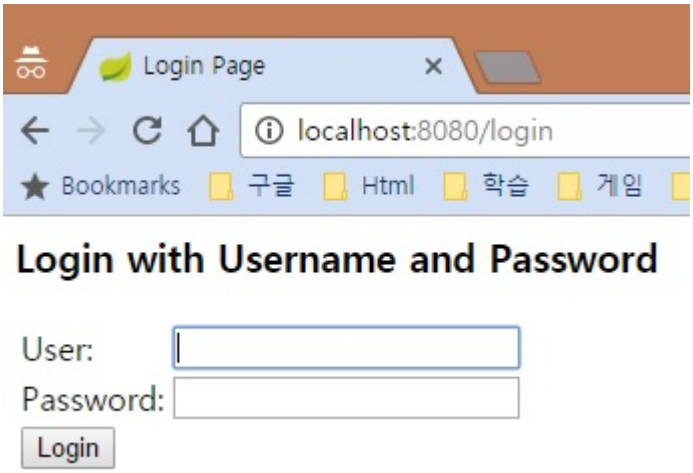
```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/spring/*.xml
  </param-value>
</context-param>

<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
```

```
class>
</listener>
```

Running Secure web application

After running your web server and visit ***index.jsp*** you will be see the default login page that generated by spring security. Because you are not authenticated.



You can login

```
username : stackoverflow
password : pwd
```

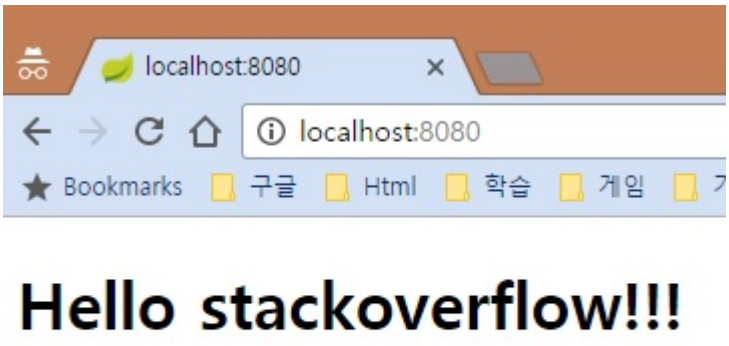
> **Note:** username and password setting on ***WEB-INF/spring/security.xml***

Displaying user name

Adding jstl tag after the "Hello", that print the username

index.jsp

```
<h1>Hello <c:out value="${pageContext.request.remoteUser}" />!!</h1>
```



Logging out

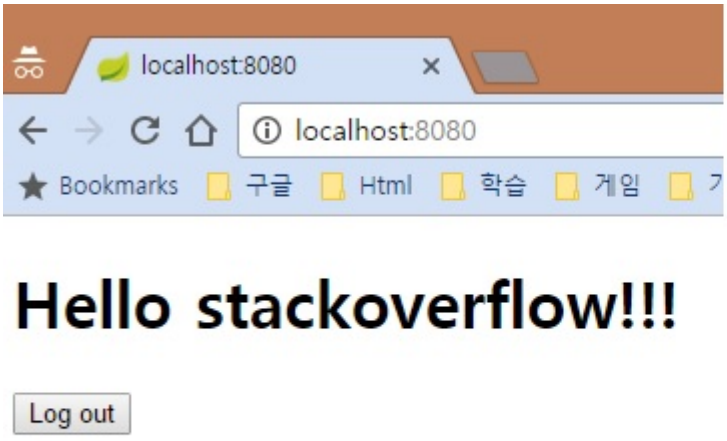
index.jsp

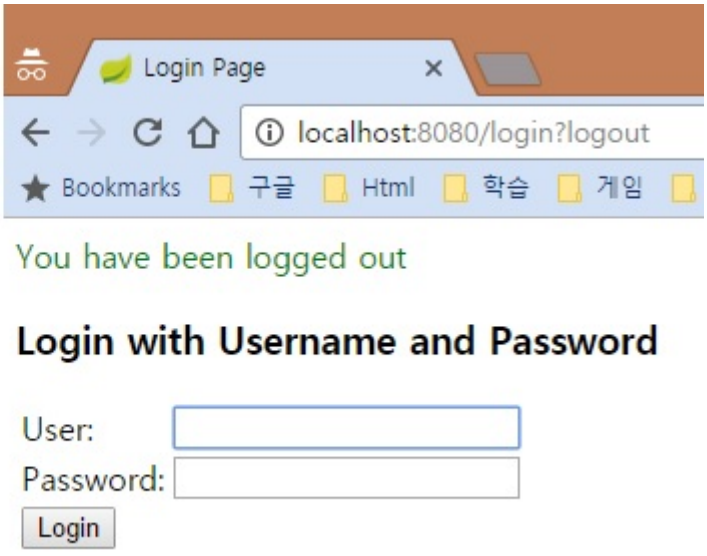
Adding form, input tags after "Hello user name", that submitting generated logging out url ***/logout*** from spring security.

```
<h1>Hello <c:out value="${pageContext.request.remoteUser}" />!!</h1>

<form action="/logout" method="post">
  <input type="submit" value="Log out" />
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
</form>
```

When you successfully log out, you see the auto generated login page again. Because of you are not authenticated now.





Read Getting started with spring-security online: https://riptutorial.com/spring-security/topic/1434/getting-started-with-spring-security

Chapter 2: Spring Security config with java (not XML)

Introduction

Typical database backed, annotation base spring security setup.

Syntax

1. configureGlobal() configure the auth object.
2. The later two SQLs may be optional.
3. configure() method tells spring mvc how to authenticate request
4. some url we do not need to authenticate
5. others will redirect to /login if not yet authenticated.

Examples

Basic spring security with annotation, SQL datasource

```
@Configuration
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
DataSource dataSource;

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth)
        throws Exception {
     auth.jdbcAuthentication().dataSource(dataSource)
    .passwordEncoder(new BCryptPasswordEncoder())
    .usersByUsernameQuery("select username,password, enabled from users where username=?")
    .authoritiesByUsernameQuery("select username, role from user_roles where username=?");

}
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.authorizeRequests().antMatchers(".resources/**", "/public/**")
            .permitAll().anyRequest().authenticated().and().formLogin()
            .loginPage("/login").permitAll().and().logout().permitAll();

    }

}
```

Read Spring Security config with java (not XML) online: https://riptutorial.com/spring-security/topic/8700/spring-security-config-with-java--not-xml-

Chapter 3: Spring Security Configuration

Examples

Configuration

Here is the corresponding Java configuration:

Add this annotation to an `@Configuration` class to have the Spring Security configuration defined in any `WebSecurityConfigurer` or more likely by extending the `WebSecurityConfigurerAdapter` base class and overriding individual methods:

```
@Configuration
@EnableWebSecurity
@Profile("container")
public class XSecurityConfig extends WebSecurityConfigurerAdapter {
```

inMemoryAuthentication
It defines an in memory authentication scheme with a user that has the username "user", the password "password", and the role "ROLE_USER".

```
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
     auth
         .inMemoryAuthentication()
             .withUser("user")
             .password("password")
             .roles("ROLE_USER");
  }

  @Override
  public void configure(WebSecurity web) throws Exception {
     web
         .ignoring()
             .antMatchers("/scripts/**","/styles/**","/images/**","/error/**");
  }
```

HttpSecurity
It allows configuring web based security for specific HTTP requests. By default it will be applied to all requests, but can be restricted using `requestMatcher(RequestMatcher)` or other similar methods.

```
  @Override
  public void configure(HttpSecurity http) throws Exception {
     http
         .authorizeRequests()
             .antMatchers("/rest/**").authenticated()
             .antMatchers("/**").permitAll()
             .anyRequest().authenticated()
             .and()
         .formLogin()
             .successHandler(new AuthenticationSuccessHandler() {
```

```
                @Override
                public void onAuthenticationSuccess(
                    HttpServletRequest request,
                    HttpServletResponse response,
                    Authentication a) throws IOException, ServletException {
                        // To change body of generated methods,
                        response.setStatus(HttpServletResponse.SC_OK);
                }
            })
            .failureHandler(new AuthenticationFailureHandler() {
                @Override
                public void onAuthenticationFailure(
                    HttpServletRequest request,
                    HttpServletResponse response,
                    AuthenticationException ae) throws IOException, ServletException {
                        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                }
            })
            .loginProcessingUrl("/access/login")
            .and()
        .logout()
            .logoutUrl("/access/logout")
            .logoutSuccessHandler(new LogoutSuccessHandler() {
                @Override
                public void onLogoutSuccess(
                    HttpServletRequest request,
                    HttpServletResponse response,
                    Authentication a) throws IOException, ServletException {
                        response.setStatus(HttpServletResponse.SC_NO_CONTENT);
                }
            })
            .invalidateHttpSession(true)
            .and()
        .exceptionHandling()
        .authenticationEntryPoint(new Http403ForbiddenEntryPoint())
            .and()
        .csrf() //Disabled CSRF protection
            .disable();
  }
}
```

Read Spring Security Configuration online: https://riptutorial.com/spring-security/topic/6600/spring-security-configuration

Credits

S. No	Chapters	Contributors
1	Getting started with spring-security	Alex78191, AMAN KUMAR, Community, dur, Gnanam, kartik, Panther, sayingu, Xtreme Biker
2	Spring Security config with java (not XML)	Maxi Wu
3	Spring Security Configuration	dur, ojus kulkarni