Spring Boot Security SIG
LABS

Wat heb je nodig:
– Java 8+ (het liefste 11)
– een IDE, als voorbeeld gebruiken wij Intellij

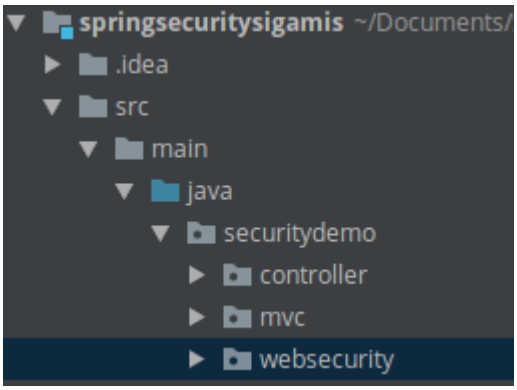
In deze labs gaan we een basis authenticatie maken met Spring Security. Je leert met security structuren en annotations werken die juist zo kenmerkend zijn voor Spring.

De labs is voor sommige misschien wat eenvoudiger, dat is niet zonder reden. Wij willen dat iedereen de concepten begrijpt achter authenticatie en autorisatie, niet de use-case afhankelijke specifieke implementatie.

Stap 1
Maak een nieuw SpringBoot project aan via de Spring Initialzr website (https://start.spring.io/). Je mag zelf je artifact en groupid kiezen.

Als het goed is heb je nu een leeg project met een pom.xml.

Stap 2
Maak in de Java folder de volgende packages aan.



Stap 3
We gaan ervoor zorgen dat de *pom.xml* de juiste dependencies heeft. Dit is erg makkelijk, maar weet je ook wat je doet? Hier eerst wat korte vragen die goed zijn om te beantwoorden.

Wat is het verschil tussen een groupid en een artifactid?

_____
_____
_____

Binnen de pom.xml heb je naast <dependencies> ook een <parent>, wat betekent dit precies?

_____
_____
_____
_____

Wat doet de <build> binnen de pom.xml?

_____
_____
_____
_____

Als je al deze vragen hebt beantwoord kan je naar de volgende vraag.

Stap 4
We gaan nu daadwerkelijk iets in de pom.xml zetten. Maar we gaan niet copy-pasten!
– In de <*parent*> tag moet staan:

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.4.RELEASE</version>
```

– In de <*dependencies*> moeten de volgende dependencies zitten.
spring-boot-starter-thymeleaf
spring-boot-starter-web
spring-boot-starter-security
spring-boot-starter-test (let op de scope!)
spring-security-test (let op de scope!)

Zoek ze in https://mvnrepository.com

Stap 5
Zet de <*properties*> in de pom.xml. We hoeven alleen de *java.version* aan te geven. Zet deze naar 11.

Stap 6
Genoeg gezocht. We hoeven alleen nog maar de <*build*> te definiëren. Deze is gratis!

```
<build>
   <plugins>
      <plugin>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
   </plugins>
</build>
```

Als het goed is heb je nu je *pom.xml* helemaal af, en je begrijpt precies wat het allemaal betekend! :)

Stap 7
Omdat we niet de hele tijd in Postman willen zitten moeten we een aantal webpagina's hebben. Maak in de *resources* folder een package genaamd "templates" aan. In deze folder zetten we de volgende bestanden. Download of kopieer ze.

https://github.com/samvruggink/sig-htmlpages

Dit zijn Thymeleaf files. Oftewel een server-side java template engine. We gaan hier verder niet op in. Lees wel even alle HTML bestanden goed door wat ze doen, of wat je denkt dat ze doen.

Stap 8
We gaan terug naar de java folder met onze packages. Maak een class aan genaamd *MvcConfig* onder de package *mvc*. Hier voegen we onze Thymeleaf files toe aan de applicatie, we zetten een URL path en geven de naam van de resource mee.

We beginnen boven aan de klasse met de nodige imports.
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

Er staat nu nog alleen *public class MvcConfig.* Zorg ervoor dat deze klasse:
- Als Configuration wordt ingesteld via annotatie
- *WebMvcConfigurer* implementeert.

Nu gaan we aan de een methode maken die *addViewControllers* heet. Deze heeft 1 argument namelijk *(ViewControllerRegistry registry).*

Voeg de onderstaande code toe. Deze linkt een URL aan een van onze Thymeleaf templates.
registry.addViewController("/home").setViewName("home");
registry.addViewController("/").setViewName("home");
registry.addViewController("/succes").setViewName("succes");
registry.addViewController("/login").setViewName("login");

We hebben nu succesvol de MvcConfig klasse gemaakt.

Stap 9
Nu gaan we de WebSecurityConfig maken. In de package van websecurity maak een klasse die *WebSecurityConfig* heet en import het volgende.

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;

Zoals de naam al zegt is dit een configuratieklasse zodat we onze webpaginas kunnen afschermen.

Neem de volgende methode over en zet deze in de klasse.
https://pastebin.com/bkRj9LcT

@EnableWebSecurity zorgt ervoor dat we een soepele MVC integratie hebben en natuurlijk Spring Security gebruiken.

Nog wat vragen:

Welk pad heeft geen authenticatie nodig?

_____
_____
_____

_____

Wat gebeurt er met pagina's die we niet hier in hebben opgenomen?

_____
_____
_____
_____

Waar wordt een user naartoe gestuurd wanneer deze succesvol inlogt?
o Specifieke pagina
o Is nog niet aangegeven
o De pagina die van tevoren is opgevraagd waar authenticatie voor nodig was
o Geeft momenteel een error terug

Stap 10
Nu gaan we een user maken. Omdat we niet 2 uur bezig willen zijn met een MySQL connectie opzetten maken we gebruik van een simpele Bean.

Wat is een Bean in Java?

_____
_____
_____
_____

Voeg deze methode toe aan de WebSecurityConfig

```
@Bean
  @Override
  public UserDetailsService userDetailsService() {
    UserDetails user =
      User.withDefaultPasswordEncoder()
        .username("VOEGHIERJEUSERNAMEIN")
        .password("JEWACHTWOOORDDD")
        .roles("USER")
        .build();

    return new InMemoryUserDetailsManager(user);
  }
```

Vergeet niet je eigen username en password te kiezen. We slaan deze in memory op om snel verder te gaan met de labs.

Stap 11
Nu gaan we in de controller een *AccountController* klasse aanmaken, natuurlijk in de package *controller*. Deze is vrij simpel en met jullie Spring kennis kunnen jullie dit met een beetje Googlen zelf.

Requirements:

- Het is een @Controller
- Zorg ervoor dat de AccountController een requestmapping heeft van "/account".

Maak twee functies met de goede HTTPMapping
- GET – transferMoney() : String
- POST (/transfer) – transfer() : String

transfermoney geeft “success” terug.
Transfer geeft “transfersuccess” terug.

Stap 12
Maak nu een main klasse aan genaamd *Application* in de *securitydemo* package.
Controleer of de volgende code aanwezig is, anders, voeg het toe:

```
import org.springframework.boot.SpringAimport org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

   public static void main(String[] args) throws Throwable {
      SpringApplication.run(Application.class, args);
   }
}
```

Wat doet @SpringBootApplication en hoe zorgt dit ervoor dat de totaaloplossing werkt?

_____
_____
_____
_____

Stap 12
Snap je hoe de code werkt? Wat er gebeurt bij het inloggen? Kan je voorspellen wat er gebeurt. Probeer dit eens.

Stap 14
Omdat we nu alles hebben om de applicatie te draaien, ga naar de folder van je project en gebruik het volgende commando.
mvn clean install

Als het goed is krijg je een build succes.

Wat betekent de clean in “mvn clean install”?

_____
_____
_____
_____

Stap 15
Ga naar je target folder en gebruik het volgende commando:
java –jar [NAAM]-1.0-SNAPSHOT.jar

Waarom genereert Maven een SNAPSHOT? Wat geeft dat precies aan?

_____
_____
_____
_____

Stap 16
Nu je de applicatie hebt draaien kan je naar je localhost gaan op poort 8080. Probeer eens:
– Inloggen met verkeerde username/password
– Inloggen met goede username/password

We hebben het helemaal niet over bruteforcing attacks gehad. Kan jij een scenario bedenken waar een account-lockout met bruteforcing een groot probleem is?

_____
_____
_____
_____

Je hebt nu succesvol een eenvoudig voorbeeld van Spring Security gebouwd.

Stap 17
Weet je nog de csrf.disable die ergens stond. Die gaan we nu gebruiken. Je bent nog steeds ingelogd en openen nu de file “attacker.html” in je browser. Ga eerst even kijken wat de file precies doet.

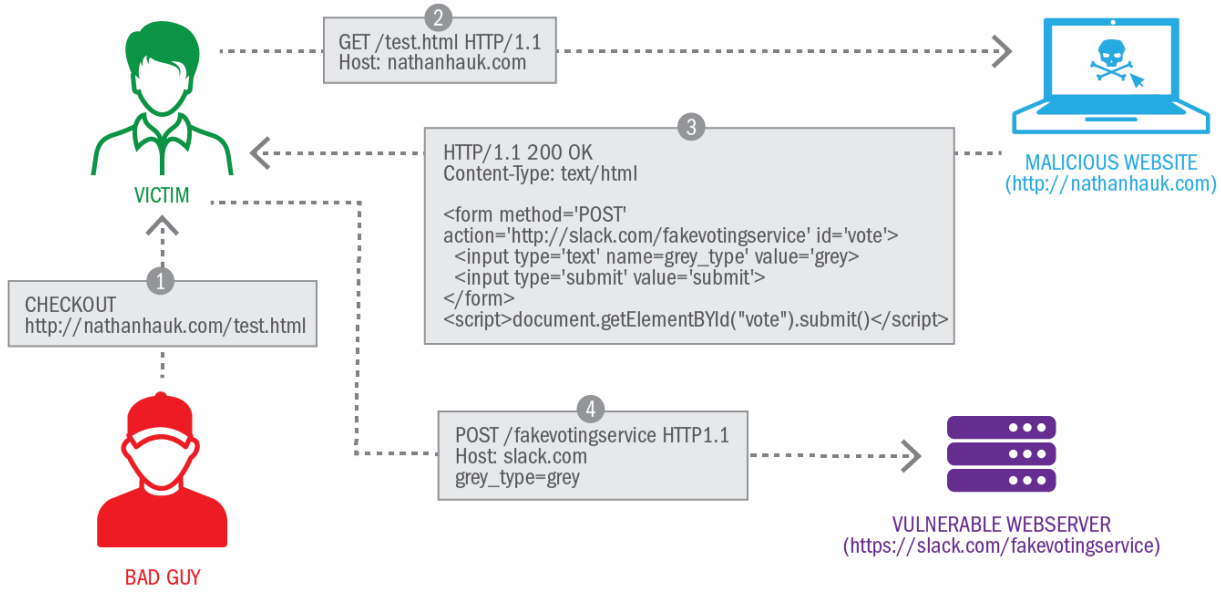
Stap 18
Even op de GIF na, we gaan nu hacken! Wauw.
Links onder de GIF staat een klik “elite hackzorz button”, klik daar eens op.

..
..

…

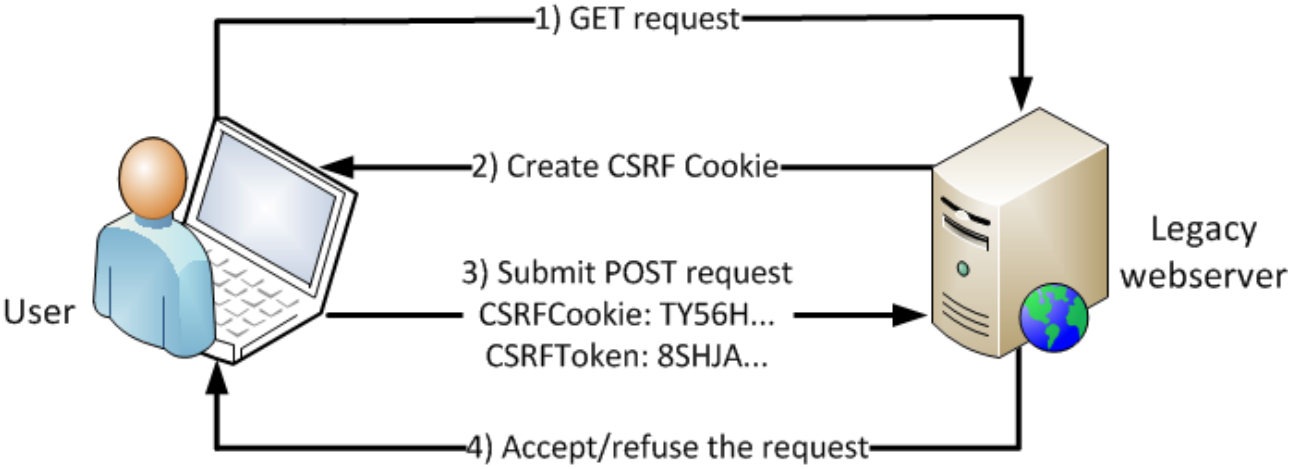
Bedankt, fijne dag he!

Stap 19
Wat is er nou precies gebeurt? Je bent op de pagina /account/transfer gekomen door CSRF.



Dit plaatje illustreert wat er aan de hand is. Attacker.html stuurt een POST request vanaf een ander domein naar onze server. Dat is natuurlijk niet te bedoeling, we zijn inderdaad ingelogd en hebben een valide cookie in onze browser. Hoe los je dit nou op?

Stap 20



Dit is de oplossing. Een random id die telkens word meegestuurd. Zo kan je attacker nooit weten wat het ID is. Haal nu de http.csrf().disable; uit de code (WebSecurityConfig.java).

Gebruik opnieuw mvn clean install
Run de jar
Log in
En probeer nog een keer attacker.html uit te voeren. Wat gebeurt er nu?

Stap 42
Bedankt, wil je meer weten over spring security.

https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

Wil je meer tutorials volgen met zaken zoals roles en privileges, anti brute force authentication attemps, session management en meer. Volg ze hier! Erg leerzaam en leuk.

https://www.baeldung.com/security-spring