Security

SPRING SECURITY & JEE 8 SECURITY

Spring Security är väldigt kraftfull, enkel att konfigurera och bygga på med nya saker.

Spring Security stödjer OAuth, OAuth2, SAML och Kerberos.

Den bygger på Authentication och Authorization för att säkerställa att rätt användare kommer på rätt end points.

I en webb applikation är back end security ett måste, då JavaScript alltid kan hackas och tas sig förbi eventuella guards.

När en request kommer utifrån triggas en *Authentication Mechanism* som samlar in credentials.

Denna mekanism består av flera delar.
Authentication Request, ett objekt skapas.
Detta objekt skickas till *Authentication Manager*.
Managern ansvarar för att skicka objektet genom en kedja av *Authenication Providers.*

Providern kommer fråga UserService att skapa ett UserObjekt.
Detta UserObjekt, kommer sen att få alla detaljer som tillhör denna specifika användare.

När *Authentication* objektet är fyllt returneras det till *Authentication Mechanism*. Om det bedöms som OK sänds förfrågan on inloggning till *Security Context Holder*, och görs om.

Om man däremot inte uppfyller OK, skickas felet 403 Unauthorized tillbaka till användaren för att försöka igen. Ibland med hjälpfullt meddelande som att lösenordet är fel eller e-post adressen inte stämmer.

Felet genereras med hjälp av *AbstractSecurityInterceptor* och *ExceptionTranslationFilter*.

För att implementera Security måste man konfigurera det. Vanligvis delas detta upp i olika klasser.

WebSecurity skapar ett Servlet Filter som beskyddar URL, validerar credentials m.m.

```
@EnableWebSecurity
public class WebSecurityConfig { ..... }
```

Oftast så handhar den en endpoint, där alla kan komma åt, och som enbart kommer testa om du har tillstånd, vilken roll du har, skicka tillbaka eventuella tokens, beskydda databasen från CSRF m.m

Nästa steg är att implementera *AbstractSecurityWebApplicationInitilizer*.

Normalt om man bygger en Spring eller SpringBoot så blir WebSecurity automatiskt registrerat men annars får man göra detta manuellt i en seperat klass.

HTTP SECURITY

Nästa steg är att implementera HttpSecurity.

Skillnad Web Security och HTTP Security?

Web Security tillhandahåller *hur* man autentisera sig.
Http Security håller koll på att alla autentiserade faktiskt har tillstånd till eventuella endpoints, och vad de får göra.

I WebSecurityConfigurerAdapter finns en metod som heter
configure(HttpSecurity http) throws Exception { ….. }

Denna konfigurationsmetod börjar alltid med *.http* och tar därefter in flertalet påbyggnadskrav.

.addFilter() lägger man in sin Authentication Manager i, följt av en *.antMatchers(HttpMethod.POST)* som hänvisar alla logins till rätt endpoint.

Därefter så säkras *alla* endpoints med t.
ex. *.antMatchers("/path").hasRole("USER").*

Man kan om man vill ta bort cors och csrf fel i denna metod, men det avråds starkt. Låt Security ta hand om vilka metoder som har tillträde till applikationen.

JEE8 Security

I JEE7 och tidigare fick all konfiguration göras i web.xml.
JEE8 Security API har fått ett helt nytt *HttpAuthenticationMechanism* Interface.

Det finns tre fördefinerade metoder som kan triggas med annoteringar.

@BasicAuthenticationMechanismDefinition
@FormAuthenticationMechanismDefinition
@CustomFormAuthenticationMechanismDefinition

Alla metoder följer Servlet Specifikationen för inloggning.

Med detta menas att man har en *Authentication* och en *Authorization* implementation som ligger i en egen container.

Då olikt SpringBoot, JEE styrs med *EJB* bönor, måste man anropa *Security Context* med en *CDI*.

Detta görs med:

```
@Inject
private SecurityContext securityContext;
```

Efter detta kan man konfigurera vilka endpoints och metoder en användare har access till med enkla boolean sparade variabler.

Man kan även annotera med *@ServletSecurity*, *@HttpConstraints*, *@HttpMethodConstraints*. Detta är likt Spring Securitys HttpSecurity metod. Varje anrop kommer då checkas av och säkerställas att den användaren har tillträde till endpointen.

Identity Store

JEE servern bör tillhandahålla två typer av Identity Stores:

Databas eller LDAP (Lightweight Directory Access Protocol).

Man måste ha JNDI data-source till den externa databasen och metoden som ansvarar för kontrollen annoteras med: *@DatabaseIdentityStoreDefinition*

LDAP metoden i sin tur annoteras med *@LdapIdentityStoreDefinition*

Båda metoderna skall skicka med konfigurationsdata, dels för var IdentityStore finns, dels grupper och filter.

Vill man kan man även göra sin egna IdentityStore.

Kortfattat kan man säga att JEE 8 Security kan både vara jobbigare och lättare än Spring Security. Kan man använda de fördefinerade implementationerna går det snabbt att få igång säkerheten. Skall man konfigurera själv tar det lite längre tid då det handlar om EJB Containers.

JWT och Security

Vad många tror är att Security och JWT är samma sak. Det är inte det. I Security räcker det med att parametern "role" kommer med och har en användbar roll. Vill man koppla in JWT, så måste man konfigurera in detta i *Authorization* delen.

JWT skall sedan sättas in i Response Headern och när man gör ett anrop med t.ex. Insomnia, kan man läsa ut token raden i Header sektionen och sen måste man sätta in denna token i varje anrop under "*Authentication*".