

# Spring Security Özelleştirmeleri (İleri Düzey)

# Spring Security ve JWT Entegrasyonu

- REST servis çağrılarının kimliklendirilmiş ve uygun yetkiye sahip istemciler tarafından çağrılmasını sağlayan “**access token**” kabiliyetidir
- Başarılı **kimliklendirme sonrası** istemciye gönderilmek üzere bir **JWT access token** üretilir
- İstemci müteakip isteklerinde bu token’ı **Authorization request header** değeri olarak gönderir

# Spring Security ve JWT Entegrasyonu

- Sunucu tarafında Spring Security JWT token'ı **header'dan alır, validate eder ve kullanıcı bilgisini** de token'ın içerisinden extract eder
- Ardından da bu kullanıcının current request'in yapıldığı resource üzerinde **yetkisi olup olmadığının** kontrolü gerçekleştirilir

# Spring Security ve JWT Entegrasyonu

- Spring Security JWT entegrasyonu için hem **kimliklendirme** hem de **yetkilendirme** tarafı için ilgili **Filter**'larda **değişiklik yapmak** gerekir
- Kimliklendirme tarafında **UsernamePasswordAuthenticationFilter** sınıfı extend edilerek **attemptAuthentication** metodu içerisinde JSON formatındaki credentials bilgileri extract edilerek kimliklendirme işlemi gerçekleştirilir

# Spring Security ve JWT Entegrasyonu

- Başarılı kimliklendirme sonrası da istemciye gönderilecek olan JWT token'ını oluşturmak için **successfulAuthentication** metodu override edilmelidir
- Müteakip isteklerde JWT token'ı validate etmek için ise **BasicAuthenticationFilter** sınıfı extend edilerek **doFilterInternal** metodu override edilmelidir
- Kısaca bu metot içerisinde JWT token validate edilir ve Authentication token nesnesi oluşturulur

# Spring Security ve JWT Entegrasyonu

```
<beans>
  <bean id="jwtAuthFilter" class="x.y.z.JwtAuthenticationFilter">
    ...
  </bean>

  <bean id="jwtAuthzFilter"
        class="x.y.z.JwtAuthorizationFilter">
    ...
  </bean>

  <security:http>
    ...
    <security:custom-filter ref="jwtAuthFilter"
                           before="FORM_LOGIN_FILTER"/>

    <security:custom-filter ref="jwtAuthzFilter"
                           before="BASIC_AUTH_FILTER"/>
  </security:http>
</beans>
```

# FilterChainProxy'nin Explicit Konfigürasyonu

- Spring Container içerisinde en az bir http elemanı tanımlanması  
**springSecurityFilterChain** isminde ve **FilterChainProxy** tipinde bir bean yaratılmasını tetikler
- Birden fazla http elemanının her biri de bu bean altında ayrı ayrı **SecurityFilterChain** bean'leri şeklinde ele alınmaktadır

# FilterChainProxy'nin Explicit Konfigürasyonu

- **FilterChainProxy** bean'i, hiç **security:http** elemanı olmadan normal **bir bean** şeklinde de tanımlanabilir

```
<bean id="filterChainProxy" class="org.springframework.security.web.FilterChainProxy">
  <constructor-arg>
    <list>
      <security:filter-chain pattern="/mvc/**"
        filters="securityContextPersistenceFilterWithASCFALSE,
        basicAuthenticationFilter,exceptionTranslationFilter,
        filterSecurityInterceptor" />
      <security:filter-chain pattern="/**"
        filters="securityContextPersistenceFilterWithASCTrue,
        formLoginFilter,exceptionTranslationFilter,
        filterSecurityInterceptor" />
    </list>
  </constructor-arg>
</bean>
```

Bu durumda web.xml'deki DelegatingFilterProxy filter tanımının ismi de bean ismi ile eşleşmelidir.



# SecurityMetadataSource'un Özelleştirilmesi

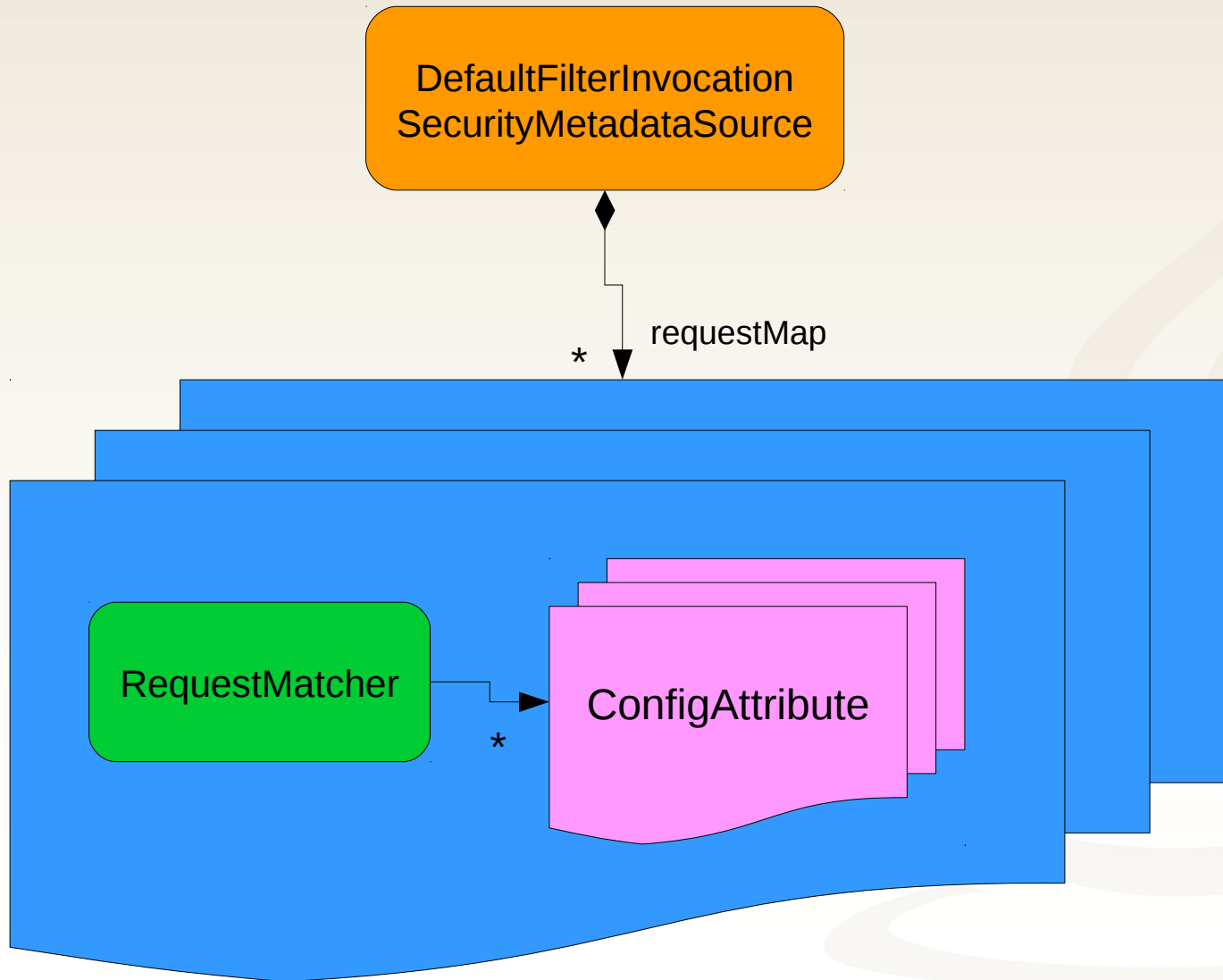
- **security:http** elemanı kullanılmayan durumlarda genellikle **FilterSecurityInterceptor** bean'ine enjekte edilen **SecurityMetadataSource'un** da özelleşmesi söz konusu olmaktadır

```
<bean id="filterSecurityInterceptor"  
  class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">  
  
  <property name="authenticationManager" ref="authenticationManager" />  
  <property name="accessDecisionManager" ref="accessDecisionManager" />  
  <property name="securityMetadataSource">  
    <security:filter-security-metadata-source use-expressions="false">  
      <security:intercept-url pattern="/**" access="IS_AUTHENTICATED_FULLY" />  
    </security:filter-security-metadata-source>  
  </property>  
</bean>
```

# SecurityMetadataSource'un Özelleştirilmesi

- Ya da **intercept-url** tanımlarının **veritabanı** gibi bir yerde yönetilmesi söz konusu olabilir
- Bu durumda **DefaultFilterInvocationSecurityMetadataSource** sınıfından bir bean oluşturularak içeriği DB'den elde edilmelidir
- Daha sonra bu bean **FilterSecurityInterceptor** bean'ine enjekte edilmelidir

# SecurityMetadataSource'un DB Entegrasyonu



# SecurityMetadataSource İçin Örnek Veri Modeli

REQUEST\_MATCHERS\_TABLE

ID	ORDER	REQUEST_MATCHER_PATTERN
1	0	/vets.jsp
2	1	/createVet.jsp
3	2	/**

CONFIG\_ATTRIBUTES\_TABLE

ID	PATTERN_ID	CONFIG_ATTR_VALUE
1	1	ROLE_USER
2	1	ROLE_EDITOR
3	2	ROLE_EDITOR
4	3	IS_AUTENTICATED_FULLY

# RequestMatcher ve ConfigAttribute Arayüzleri

- **RequestMatcher** arayüzü için  
AntPathRequestMatcher,  
RegexRequestMatcher gibi sınıflar  
kullanılabilir
- **ConfigAttribute** arayüzü için  
SecurityConfig veya  
WebExpressionConfigAttribute sınıfları  
kullanılabilir

# RequestMatcher Örnekleri

- `/**` gibi bir RequestMatcher değeri **AntPathRequestMatcher** ile ifade edilir
- `/find.*` gibi bir değer ise **RegexRequestMatcher** ile ifade edilir
- Herhangi bir request URI ile eşleşmesi için **AnyRequestMatcher** kullanılabilir

# ConfigAttribute Örnekleri

- `ROLE_USER`, `IS_AUTHENTICATED_FULLY` gibi değerler **SecurityConfigAttribute** ile ifade edilirler
- `hasRole('ROLE_USER')`, `permitAll`, `isFullyAuthenticated()` gibi SpEL değerleri ise **WebExpressionConfigAttribute** ile ifade edilirler
- **SpelExpressionParser** ile `WebExpressionConfigAttribute`'un beklediği `Expression` nesne oluşturulabilir

# İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- [info@java-egitimleri.com](mailto:info@java-egitimleri.com)

