Spring Security ve Vaadin Entegrasyonu

harezmi bilişim çözümleri

Spring Security ve Single Page Ajax Uygulamaları

JAVA Eğitimleri

- Spring Security Servlet **Filter tabanlı bir framework**'tür
- Web kaynaklarının güvenlikli hale getirilmesinde klasik **sayfalar arası geçişe sahip web uygulamaları** baz alınmıştır
- Kendine ait bir ajax iletişim mekanizması olan **single page ajax uygulamaları** içerisinde web kaynaklarının yetkilendirilmesi çok işe yaramamaktadır

Spring Security ve Single Page Ajax Uygulamaları

- Bu tür uygulamalarda **ilk sayfa yüklendikten sonra** aynı sayfanın içerisinde bölümler bu iletişim kanalı üzerinden dinamik olarak güncellenmekte veya değişmektedir
- Dolayısı ile bu değişiklik ve güncellemelerin gerçekleştirildiği yerlerde doğrudan Spring Security'nin **yetkilendirme katmanına erişerek** yetki denetimi yapılabilir

Spring Security ve Vaadin

- Spring Security, Vaadin ile geliştirilen web uygulamalarında **kimliklendirme** ve **yetkilendirme** ihtiyaçlarını karşılamak için kullanılabilir
- View arayüzlerinde **Navigator ile dolaşırken** güvenlik kontrolü yapmak için Vaadin Spring Addon'undan da yararlanılabilir

Vaadin ve Kimliklendirme

- Kimliklendirme işleminde **UsernamePasswordAuthenticationProcessingFilter**'ın **HTTP POST** metodu ile yapılmış bir **FORM submission**'ını işlemesi gerekir
- Dolayısı ile **login arayüzü**nü Vaadin içerisinde yapmak yerine **ayrı bir JSP veya HTML sayfası** yaparak Vaadin uygulamasına giriş sağlamaka en kolay yaklaşım olacaktır

Spring Security Vaadin Konfigürasyonu

```
<web-app...>
     <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>/app/*</url-pattern>
     </filter-mapping>

     <servlet>
          <servlet-name>VaadinServlet</servlet-name>
          <servlet-class>com.vaadin.spring.server.SpringVaadinServlet</servlet-class>
          <init-param>
               <param-name>UI</param-name>
               <param-value>com.javaegitimleri.petclinic.view.PetclinicUI</param-value>
          </init-param>
     </servlet>
     <servlet-mapping>
          <servlet-name>VaadinServlet</servlet-name>
          <url-pattern>/app/*</url-pattern>
     </servlet-mapping>

     <servlet-mapping>
          <servlet-name>VaadinServlet</servlet-name>
          <url-pattern>/VAADIN/*</url-pattern>
     </servlet-mapping>
</web-app>
```

VaadinServlet'in url-pattern'ı intercept edilmelidir

Vaadin tema resource'larına erişimde Spring Security'nin devreye girmesine gerek yoktur

Spring Security Vaadin Konfigürasyonu

```
<beans...>
    <security:http pattern="/**" >

        <security:form-login
                login-page="/login.jsp"
                authentication-failure-url="/login.jsp?error=true"/>

        <security:intercept-url   pattern="/login.jsp"
                                  access="permitAll"/>
        <security:intercept-url
                pattern="/**" access="isAuthenticated()" />

        ...
    </security:http>
</beans>
```

Login sayfası ve login sırasında hataları göstermek için bir JSP veya HTML sayfa kullanılır

Vaadin uygulaması genellikle tek bir UI'dan oluştuğu herhangi bir request'e erişim söz konusu olduğunda bu erişimden önce kimliklendirme yapılıp yapılmadığına bakmak yeterlidir. Detaylı yetkilendirme UI nesnesinin içerisindeki alt bileşenlerde yapılacaktır

Login Arayüzünün Vaadin ile Geliştirilmesi

- **Login arayüzü** Vaadin ile geliştirilen bir **View**, **UI** veya **Dialog** olabilir
- Bu durumda VaadinServlet'in devreye girmesini sağlayan **uri-pattern** altında kalan **UIDL, HEARTBEAT** gibi URI'lar herkes tarafından erişilebilir olmalıdır

```
<security:http>
    <security:intercept-url pattern="/**/UIDL/**" access="permitAll"/>
    <security:intercept-url pattern="/**/HEARTBEAT/**" access="permitAll"/>
    <security:intercept-url pattern="/**/APP/PUBLISHED/**"
                                                    access="permitAll"/>
</security:http>
```

Login Arayüzünün Vaadin ile Geliştirilmesi

- Vaadin Login ekranından tetiklenen login işleminde **UsernamePassword AuthenticationFilter** kendiliğinden devreye girmeyecektir
- Kimliklendirme işlemini tetiklemek için **UsenamePassword AuthenticationFilter.doFilter()** metodunda yapılan adımları Vaadin uygulamasında da implement ederek login işlemi gerçekleştirilebilir

Login Arayüzünün Vaadin ile Geliştirilmesi

- Diğer yöntem Vaadin uygulamasından **UsernamePasswordAuthenticationFilter .doFilter()** metodunu Vaadin bileşeni içinden çağırarak kimliklendirme işlemini tetiklemektir
- Burada **current HTTP request ve response** nesnelerine ihtiyaç vardır
- Bir **ServletRequestListener** yazarak **ThreadLocal** değişkenler üzerinden bu nesnelere erişim sağlanabilir

Login Arayüzünün Vaadin ile Geliştirilmesi

```
public void authenticate(String username, String password, boolean rememberMe) {
    try {
        HttpServletRequestWrapper req = new HttpServletRequestWrapper(currReq) {
            @Override
            public String getParameter(String name) {
                if ("username".equals(name)) {
                    return username;
                } else if ("password".equals(name)) {
                    return password;
                } else if ("remember-me".equals(name)) {
                    return Boolean.toString(rememberMe);
                } else {
                    return super.getParameter(name);
                }
            }
        };
        authProcessingFilter.doFilter(req, currResp, null);
    } catch (Exception ex) {
        Notification.show("Authentication failure",Type.ERROR_MESSAGE);
    }
}
```

Vaadin ve CSRF Koruması

- Vaadin'in **kendine ait bir CSRF koruması** vardır
- Dolayısı ile **Spring Security'nin CSRF koruması** Vaadin uygulamasına gelen istekler için **devre dışı bırakılmalı**dır

```
<beans...>
    <security:http pattern="/**">
    ...
        <security:csrf disabled="true" />
    </security:http>
</beans>
```

Logout İşlemi

- Vaadin uygulaması içerisinde **logout-url**'i için bir **Link bileşeni eklemek yeterli** olacaktır

```
ExternalResource logoutResource = new ExternalResource("/petclinic/logout");
Link logoutLink = new Link("Logout", logoutResource);
```

```
<beans...>
    <security:http pattern="/**" >
        <security:logout
                logout-url="/logout"
                logout-success-url="/app" />
        ...
    </security:http>
</beans>
```

UI Fragmanlarının Yetkilendirilmesi

- Vaadin View nesneleri tarayıcı adres çubuğundan **#!viewName** ile **sayfa reload olmadan** erişilmektedir
- Dolayısı ile bu View nesnelerine erişimi **intercept-url ile** yetkilendirmek **mümkün değil**dir
- **ViewChangeListener** arayüzü implement edilerek View'a navigate edilmeden evvel yetki kontrolü yapılabilir

ViewChangeListener ile UI Fragmanlarının Yetkilendirilmesi

```
public class SecurityViewChangeListener implements ViewChangeListener {
    @Override
    public boolean beforeViewChange(ViewChangeEvent event) {
        String viewName = event.getViewName();

        Authentication authentication =
                SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null) return false;

        Collection<? extends GrantedAuthority> authorities =
                authentication.getAuthorities();

        if ("editorView".equals(viewName)) {
            return authorities.contains(
                new SimpleGrantedAuthority("ROLE_EDITOR"));
        } else if ("readerView".equals(viewName)) {
            return authorities.contains(
                    new SimpleGrantedAuthority("ROLE_EDITOR"))
                || authorities.contains(
                    new SimpleGrantedAuthority("ROLE_USER"));
        } else {
            return true;
        }
    }
    ...
}
```

ViewChangeListener ile UI Fragmanlarının Yetkilendirilmesi

- Implement edilen ViewChangeListener devreye girmesi için **Navigator nesnesine register edilmeli**dir
- Ayrıca bloklanan View sonrasında hata ekranı göstermek için bir **error ViewProvider**'da Navigator'a set edilebilir

```
Navigator navigator = new Navigator(this,content);
navigator.addViewChangeListener(new SecurityViewChangeListener());
navigator.setErrorProvider(errorViewProvider);
setNavigator(navigator);
```

Vaadin Spring Addon ile UI Fragmanlarının Yetkilendirilmesi

- ViewChangeListener implement etmek yerine **Vaadin Spring Addon**'u da kullanılabilir
- Bu addon içerisinde **ViewAccessControl** ve **ViewInstanceAccessControl** isimli iki arayüz mevcuttur
- **SpringViewProvider** herhangi bir View'a erişim öncesi bu arayüzleri implement eden Spring bean'leri ile erişim denetimi yapmayı sağlar

Vaadin Spring Addon ile UI Fragmanlarının Yetkilendirilmesi

- **ViewAccessControl**, view nesnesi elde edilmeden önce **view isminden kontrol** yapmayı sağlar
- **ViewInstanceAccessControl** ise view nesnesi elde edildikten sonra **view instance üzerinden kontrol** yapmayı sağlar
- Yetkilendirme hatası söz konusu ise yine **SpringViewProvider**'da tanımlı **accessDeniedViewClass** a karşılık gelen View nesnesi görüntülenir

Vaadin Spring Addon ile UI Fragmanlarının Yetkilendirilmesi

```
@Component
public class SecureViewAccessControl implements ViewAccessControl {

    @Override
    public boolean isAccessGranted(UI ui, String beanName) {
        Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null) return false;
        Collection<? extends GrantedAuthority> authorities =
            authentication.getAuthorities();
        if ("editorView".equals(beanName)) {
            return authorities.contains(
                    new SimpleGrantedAuthority("ROLE_EDITOR"));
        } else if ("readerView".equals(beanName)) {
            return authorities.contains(
                        new SimpleGrantedAuthority("ROLE_EDITOR"))
                    || authorities.contains(
                        new SimpleGrantedAuthority("ROLE_USER"));
            } else {
                return true;
            }
        }
}
```

Vaadin Spring Addon ile UI Fragmanlarının Yetkilendirilmesi

```
@Theme("valo")
@SpringUI
public class PetclinicUI extends UI {

    @Autowired
    private SpringViewProvider springViewProvider;

    @Override
    protected void init(VaadinRequest request) {
        VerticalLayout content = new VerticalLayout();
        Navigator navigator = new Navigator(this,content);
        navigator.addProvider(springViewProvider);
        springViewProvider.setAccessDeniedViewClass(
                                    AccessDeniedView.class);

        setNavigator(navigator);
        setContent(content);
    }
}
```

Mevcut Vaadin Spring Addon sürümünde muhtemelen bir bug dolayısı ile **accessDeniedViewClass** set edildiği takdirde SpringViewProvider herhangi bir view'a navigate edilmeye çalışıldığı vakit hata vermektedir

UI Bileşenlerinin Yetkilendirilmesi

- Vaadin UI bileşenlerine ise **URI ile** tarayıcı adres çubuğundan **page reload yapılarak erişilmekte**dir
- Dolayısı ile UI bileşenleri **intercept-url elemanı ile yetkilendirilebilir**

```
<security:http pattern="/**">
    ...
    <security:intercept-url pattern="/**/secureUI"
                            access="hasRole('ROLE_EDITOR')" />

    <security:intercept-url pattern="/**"
                            access="isAuthenticated()" />
</security:http>
```

Üst Düzey Bileşenlerinin Altındaki Bileşenlerde Yetkilendirme

- **UI, View, Window, Dialog** gibi sayfa düzeyindeki Vaadin bileşenlerinin içerisindeki Field, Table, CheckBox CustomComponent gibi alt düzey UI bileşenlerin **kullanıcı yetkisine göre görünür/görünmez veya aktif/pasif yapılması** söz konusu olabilir
- Bileşenin görünür veya aktif olması için gerekli yetki(ler), mevcut kullanıcının sahip olduğu yetkiler ile **karşılaştırılarak** ilerlenir

Üst Düzey Bileşenlerinin Altındaki Bileşenlerde Yetkilendirme

```
@SpringView(name="")
public class MainView extends CustomComponent implements View {

	private Button btn;

	public MainView() {

		btn = new Button("Click for Editor View");

		btn.addClickListener(new ClickListener() {

			@Override
			public void buttonClick(ClickEvent event) {
				UI.getCurrent().getNavigator().navigateTo("editorView");
			}
		});

		VerticalLayout content = new VerticalLayout(btn);
		content.setSpacing(true);content.setMargin(true);

		setCompositionRoot(content);
	}
	@Override
	public void enter(ViewChangeEvent event) {
		Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

		Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();

		boolean userInRole = authorities.contains(new SimpleGrantedAuthority("ROLE_EDITOR"));

		btn.setVisible(userInRole);
	}
}
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com