Spring Security

Spring security

- Spring Security is a flexible and powerful authentication and access control framework to secure Spring-based Java web application
- Spring Security allows developer to integrate security features with J2EE web application easily
- It intercepts incoming HTTP request via servlet filters, and implements “user defined” security checking
- It uses AOP for method level security
- Spring security distribution is not part Spring. It has to be downloaded separately

Major operations

- **Authentication** - process of estabilishing a principal (usually a user which can perform an action in application)
- **Authorization** - process of deciding wheather a principal is allowed to perform an action
- Authentication process establish identity of the principal, which is used for authorization decision

Spring security JAR files

- Spring Security distribution is not part of Spring bundle
- It has to be downloaded separately
- As security features can be detached anytime from the application, security configuration is placed in a separate XML file

- Following JAR files required as part of application classpath :
  - **spring-security-core-3.x.jar**
  - **spring-security-web-3.x.jar**
  - **spring-security-config-3.x.jar**

Namespace Design

- Based around the large-scale dependencies within the framework:
  - *Web/HTTP Security* – Set up filters and related service beans (authentication, secure URL's, login,...)
  - *AuthenticationManager* - requests authentication (default)
  - *AccessDecisionManager* – web and method security (default/customizable bean)
  - *AuthenticationProviders* – used by authentication manager for user authentication (namespace support/customizable beans)
  - *UserDetailsService* – related to authentication providers, used by other beans (bean)

Permissions

- Access permissions on any resource depend on Roles
- Every user should have a password and one or more roles assigned
- Resource security mapped to roles
- After authentication, spring checks whether user has a role required to access a resource

Configuration Steps

- Add security filters in web.xml
- security filters intercept all requests for web resources before the request goes to DispatcherServlet
- Spring provides DelegatingFilterProxy class for this purpose
- Spring automatically provides login page if cutom page is not available
- Configure context listener and context parameters to include secuirty.xml

Changes to web.xml

- Add a security filter to web.xml. This is our entry point for authentication and web resource based authorization

```
<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>
```

Changes to web.xml

- Specify security.xml as context parameter

```
<listener>
    <listener-class>
            org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
    /WEB-INF/spring-security.xml
    </param-value>
</context-param>
```

Configuration Steps

- Web security configured with http element in spring.xml
- Provide login page details
- provide Denied page details
- Provide denied page details

- Configure Authentication Provider
- AuthenticationProvider is responsible to handle requests for user details (password, role etc)

AuthenticationProvider

- Spring supports multiple implementations for AuthenticationProvider
  - Specifying user details in xml file itself
  - By configuring UserDetailsService object (class should implement UserDeailsservice) which has the following method
    - UserDetails loadUserByName(String name)
  - By way of storing user details in database

Security.xml

- Authentication configuration : provides details of users, roles and permissions

```
<authentication-manager>
<authentication-provider>
        <user-service>
            <user name="rod" password="rod"
                                        authorities=“supervisor,user,teller" />
            <user name="dianne" password="dianne" authorities=“teller" />
            <user name="scott" password="scott" authorities=“user" />
         </user-service>
</authentication-provider>
</authentication-manager>
```

Security.xml

- http configuration with direct role specification
- Multiple <intercept-url> elements to define access (listed order evaluated, first match used)

```
<http use-expressions="true">
    <intercept-url pattern="/index.jsp" access="permitAll" />
    <intercept-url pattern="/secure/admin/**" access="supervisor" />
     <intercept-url pattern="/listAccounts.html" access="teller" />
    <intercept-url pattern="/post.html" access="user,teller" />
    <intercept-url pattern="/**" access="denyAll" />
    <form-login />
</http>
```

Security.xml

- Authentication-provider through User-service
- User-service class should implement spring's UserDetailsService

```
<security:authentication-manager>
        <security:authentication-provider  user-service-ref="service" />
</security:authentication-manager>

  <!-- A custom service where Spring will retrieve users and their
  corresponding access levels  -->
  <bean id="service" class="spring.security.service.CustomService"/>
```

UserDetails interface

```
public interface UserDetails extends Serializable {
    Collection<GrantedAuthority> getAuthorities();
    String getPassword();
    String getUsername();
    boolean isAccountNonExpired();
    boolean isAccountNonLocked();
    boolean isCredentialsNonExpired();
    boolean isEnabled();
}
```

Configure Login & Logout

- If login page is not provided spring automatically provides login page
- URL for default login is ***/spring_security_login***
- URL for default logout is ***/j_spring_security_logout***
- We may also provide form based login to customize login page

- To customize login page the following are to be used :

    - Action : j_spring_security_check
    - Name parameter : j_username
    - Password parameter : j_password

Customized login form

- Login form JSP
- Place ${error} for login errors

```
<form action="j_spring_security_check" method="post" >

<p>
<label for="j_username">Username</label>
<input id="j_username" name="j_username" type="text" />
</p>

<p>
<label for="j_password">Password</label>
<input id="j_password" name="j_password" type="password" />
</p>
```

Customized login form

- configure login and logout

```
<http
……..
<form-login
login-page="/secure/auth/login"
        authentication-failure-url="/secure/auth/login?error=true"
        default-target-url="/secure/main/common"/>

<logout   invalidate-session="true"
        logout-success-url="/secure/auth/login"
        logout-url="/secure/auth/logout"/>

</http>
```

Passsword encoding

- Spring security uses plain text passwords by default
- Password encryption in Spring Security is encapsulated and defined by implementing *PasswordEncoder* interface
- Simple configuration of a password encoder is possible through the <*password-encoder*> declaration within the <*authentication-provider*> element

Passsword encoding

```
<authentication-manager alias="authenticationManager">
  <authentication-provider user-service-ref="jdbcUserService">
    <password-encoder  ref=“passwordEncoder”/>
  </authentication-provider>
</authentication-manager>

……….
<beans:bean
class="org.springframework.security.authentication.encoding.Md5PasswordEncoder“
 id="passwordEncoder"/>

<!--   user-service configuration  →
<security:user-service id="userDetailsService">
    <security:user name="john" password="21232f297a57a5a743894a0e4a801fc3"
authorities="ROLE_USER, ROLE_ADMIN" />
    <security:user name="jane" password="ee11cbb19052e40b07aac0ca060c23ee"
authorities="ROLE_USER" />
```

Storing encoded passwords

- Any third party tool can be used to generate MD5, MD4 orr SHA endoded passwords
- jacksum is one of the tools
- Download jacksum and install
- use jackson.jar as follows

```
java -jar jacksum.jar -a sha -q "txt:password“

java -jar jacksum.jar –a md5 -q "txt:password"
```

Method level Security

- Spring uses AOP for method security
- Provide @Secured or @PreAuthorize annotations at method level and provide roles authorised to access the method
- When a method is called during request process, spring checks the permission and sends error 405 if not authorised to access
- To enable method security use the following element

```
<security:global-method-security secured-annotations="enabled" />
```