Module 21: Spring Security

CS544: Enterprise Architecture

Spring Security

CS544 Enterprise Architecture

Wholeness

- Security, establishing who a user is (authentication), and allowing or disallowing actions (authorization) are vital to any serious application.
- In this Spring Security Module we will look at:
  - Authentication in a web environment
  - Requiring Authorization for certain web pages
  - Requiring Authorization for method calls
  - Defending against common attacks

Spring Security:

WEB SECURITY

Spring Security

- Authentication
  - Many different types authentication supported
  - Many different types of data sources supported
  - Easy to add your own

- Authorization
  - Web Security, URL patterns
  - Business Method, annotations
  - Advanced Access Control Lists (ACL) and Expressions

Web.xml configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
     id="WebApp_ID" version="2.5">
    <display-name>security</display-name>
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- Needed when using Spring with Filter -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/springconfig.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <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>
</web-app>
```

Can use with or without SpringMVC

Filter applies security

springconfig.xml

Loaded by filter

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:sec="http://www.springframework.org/schema/security" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
     beans-3.0.xsd http://www.springframework.org/schema/security
     http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    <sec:http>
        <sec:intercept-url pattern="/important.jsp" access="ROLE_USER"/>
        <sec:form-login />
        <sec:logout />
    </sec:http>

    <sec:authentication-manager>
        <sec:authentication-provider>
            <sec:user-service>
                <sec:user name="test" password="123" authorities="ROLE_USER, ROLE_ADMIN" />
                <sec:user name="bob" password="bobiscool" authorities="ROLE_USER" />
            </sec:user-service>
        </sec:authentication-provider>
    </sec:authentication-manager>
</beans>
```

Uses many security elements, everything starts with <sec:… >

springconfig.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
xmlns:beans="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
     beans-3.0.xsd http://www.springframework.org/schema/security
     http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    <http>
        <intercept-url pattern="/important.jsp" access="ROLE_USER"/>
        <form-login />
        <logout />
    </http>

    <authentication-manager>
        <authentication-provider>
            <user-service>
                <user name="test" password="123" authorities="ROLE_USER, ROLE_ADMIN" />
                <user name="bob" password="bobiscool" authorities="ROLE_USER" />
            </user-service>
        </authentication-provider>
    </authentication-manager>
</beans:beans>
```

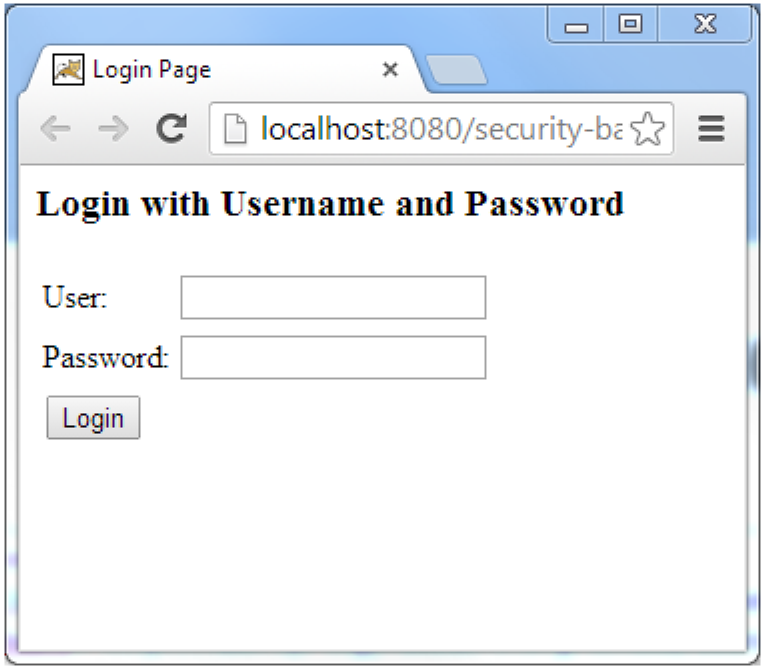
Make security the primary namespace

<http> elements specify url patterns for security

Authentication manager / provider configuration

Generated login.jsp

- Spring Security generates a form-login
  - If we don't specified a login page on <form-login>


Login Page

localhost:8080/security-ba

Login with Username and Password

User:

Password:

Login

Custom Login Form

login.jsp

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Login Page!</h1>
        <c:if test="${error eq true}">
            <p>${sessionScope["SPRING_SECURITY_LAST_EXCEPTION"].message}</p>
        </c:if>
        <form method="post" action="<c:url value='j_spring_security_check' />">
            User: <input name="j_username" value='<c:if test="${not empty param.login_error}"><c:out
     value="${SPRING_SECURITY_LAST_USERNAME}"/></c:if>'/> <br />
            Pass: <input type="password" name='j_password' /> <br />
            <input type="submit" />
        </form>
    </body>
</html>
```

springconfig.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
xmlns:beans="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
     beans-3.0.xsd http://www.springframework.org/schema/security
     http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    <http pattern="/index.jsp" security="none" />

    <http>
        <intercept-url pattern="/login.jsp" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
        <intercept-url pattern="/loginfailed" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
        <intercept-url pattern="/**" access="ROLE_USER" />

        <form-login login-page="/login.jsp" authentication-failure-url="/loginfailed"
                    default-target-url="/success"/>

        <logout logout-success-url="/index.jsp"/>
    </http>

    <authentication-manager>
        <authentication-provider>
            <user-service>
                <user name="test" password="123" authorities="ROLE_USER, ROLE_ADMIN" />
                <user name="bob" password="bobiscool" authorities="ROLE_USER" />
            </user-service>
        </authentication-provider>
    </authentication-manager>
</beans:beans>
```

Everyone can access /index.jsp, /login.jsp, and /logoutfailed

Security Config

- Multiple HTTP elements allow us to:
  - turn on / off security
- Multiple intercept-url elements on http security
  - To specify different authorization requirements
- Either turn off or allow anonymous access to:
  - login page
  - loginfailed

Otherwise it's difficult to show these pages

None VS Anonymous

- Security=“none”
  - Filter chain not applied
  - Current user data not available
- IS_AUTHENTICATED_ANONYMOUS
  - Security Filter chain is still there
  - Current user data available (may be anonymous)
  - Everyone has access

Security Tag Lib

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<!DOCTYPE html>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Tag Lib Example</title>
    </head>
    <body>
        <p>Welcome <sec:authentication property="principle.username" />,</p>
        <p>You are allowed to access:</p>
        <ul>
            <sec:authorize url="/secureArea">
                <li><a href="/secureArea">The Secured Area</a></li>
            </sec:authorize>
            <sec:authorize access="hasRole('ROLE_ADMIN')">
                <li><a href="/admin">The Admin Panel</a></li>
            </sec:authorize>
        </ul>
    </body>
</html>
```

Will only display if user is authorized to go to

Requires Security Expressions

Main Point 1

- Configuring which URL’s should be secured is quick and easy
- You can also not show links / parts of the page if the user isn’t authorized to use them
- Do less accomplish more

Spring Security:

AUTHENTICATION PROVIDERS

Authentication Providers

- So far we've just used plain text
  - Terrible for security

```
<authentication-manager>
    <authentication-provider>
        <user-service>
            <user name="test" password="123" authorities="ROLE_USER, ROLE_ADMIN" />
            <user name="bob" password="bobiscool" authorities="ROLE_USER" />
        </user-service>
    </authentication-provider>
</authentication-manager>
```

Password Encoder

- Super important:
  - Never store plain text
  - Basic hashing isn't that great either

```
<authentication-manager>
  <authentication-provider>
    <password-encoder hash="bcrypt"/>
    <user-service>
      <user name="jimi" password="d7e6351eaa13189a5a3641bab846c8e8c69ba39f"
    authorities="ROLE_USER, ROLE_ADMIN" />
      <user name="bob" password="4e7421b1b8765d8f9406d87e7cc6aa784c4ab97f"
    authorities="ROLE_USER" />
    </user-service>
  </authentication-provider>
</authentication-manager>
```

Can be: md4, md5, sha, sha-256, bcrypt

Bcrypt is recommended, it also automatically salts

Also See: org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder

JDBC Authenticator

```
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="dataSource" />
  </authentication-provider>
</authentication-manager>

<beans:bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <beans:property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <beans:property name="url" value="jdbc:mysql://localhost/cs544"/>
  <beans:property name="username" value="root"/>
  <beans:property name="password" value=""/>
</beans:bean>
```

Standard Authentication Tables

- Using the following standard schema:

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null
);
create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(username) references users(username)
);
create unique index ix_auth_username on authorities (username,authority);
```

- Inserting the username / password data:

```
Insert into users values(“test”, “123”, 1”);
Insert into users values(“bob”, “bobiscool”, 1”);
Insert into authorities values(“test”, “ROLE_USER”);
Insert into authorities values(“test”, “ROLE_ADMIN”);
Insert into authorities values(“bob”, “ROLE_USER”);
```

Non Standard Authentication Tables

```
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="dataSource"

      users-by-username-query="
        select username,password, enabled
        from users where username=?"

      authorities-by-username-query="
        select u.username, ur.authority from users u,
        user_roles ur where u.user_id = ur.user_id
        and u.username =?  " />
  </authentication-provider>
</authentication-manager>
```

http://www.mkyong.com/spring-security/spring-security-form-login-using-database/

Custom Authentication Provider

```
public class CustomAuthenticationProvider implements AuthenticationProvider {
  @Override
  public Authentication authenticate(Authentication authentication) throws AuthenticationException {
    String name = authentication.getName();
    String password = authentication.getCredentials().toString();
    if (name.equals("test") && password.equals("123")) {
      List<GrantedAuthority> grantedAuths = new ArrayList<>();
      grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));
      Authentication auth = new UsernamePasswordAuthenticationToken(name, password, grantedAuths);
      return auth;
    } else {
      return null;
    }
  }
  @Override
  public boolean supports(Class<?> authentication) {
    return authentication.equals(UsernamePasswordAuthenticationToken.class);
  }
}
```

```
<authentication-manager>
    <authentication-provider ref="customAuthenticationProvider"/>
</authentication-manager>
```

http://danielkaes.wordpress.com/2013/02/20/custom-authentication-provider-in-spring/

Multiple Authentication Providers

- Spring will try each one, top to bottom

```xml
<authentication-manager>

  <authentication-provider>
    <user-service>
      <user name="test" password="123" authorities="ROLE_USER, ROLE_ADMIN" />
      <user name="bob" password="bobiscool" authorities="ROLE_USER" />
    </user-service>
  </authentication-provider>

  <authentication-provider>
    <jdbc-user-service data-source-ref="dataSource" />
  </authentication-provider>

  <authentication-provider ref="customAuthenticationProvider" />

</authentication-manager>
```

Main Point 2

- There are many options for Authentication Providers, there are many pre-built (not discussed), others are easy to add
  - Nature of life is to grow

Spring Security:

SESSIONS AND SECURITY

Detecting Session Timeouts

- To redirect people who submit an invalid (possibly timed-out) JSESSIONID :

```
<http>
  ...
  <session-management invalid-session-url="/invalidSession.html" />
</http>
```

- After logout JSESSIONID is often still set
  - To prevent false positives:

```
<http>
  ...
  <logout delete-cookies="JSESSIONID" />
</http>
```

Concurrent Session Control

- Set how many concurrent logins are allowed:

```
<http>
  ...
  <session-management>
    <concurrency-control max-sessions="1" error-if-maximum-exceeded="true" />
  </session-management>
</http>
```

- Needs extra listener in web.xml:

```
<listener>
  <listener-class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-class>
</listener>
```

Session Fixation Protection

- By default Spring Security changes SessionID when a user logs in.
- You can control this behavior

```
<http>
  ...
  <session-management session-fixation-protection="none" />
</http>
```

- Possible values are:
  – none
  – newSession
  – migrateSession   (default Servlet 3.0 or older)
  – changeSessionId (new Servlet 3.1 / Java EE7)

XSRF Protection

- Enable Synchronizer Token Pattern:

```
<http>
  ...
  <csrf />
</http>
```

- No extra html when using Spring From tags
- Otherwise add to (all) your forms:

```
<form action="someplace" method="post">
   ...
  <input type="submit" />
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
  <-- or if you are using the spring security tag lib <sec:csrfInput />  -->
</form>
```

Adding an HTTPS Requirement

- You can force certain URLs to be HTTPS only, spring will automatically redirect

```
<http>
  <intercept-url pattern="/secure/**" access="ROLE_USER" requires_channel="https"/>
  <intercept-url pattern="/**" access="ROLE_USER" requires_channel="any"/>
  ...
</http>
```

- You can also reconfigure what ports are used

```
<http>
  ...
  <port-mappings>
    <port-mapping http="9080" https="9443" />
  </port-mappings>
</http>
```

Remember Me

- AKA, persistent-login can be used to allow automatic log in when returning

- Spring Security provides
  - Simpler hash based approach (less secure)
  - Persistent Token approach (requires datastore)

http://docs.spring.io/spring-security/site/docs/3.2.2.RELEASE/reference/htmlsingle/#remember-me

Main Point 3

- Sessions and Security
- Every action has a reaction

Spring Security:

METHOD SECURITY

Method Security

- Spring's @Secured annotation
- JSR-250's annotations
- AOP Pointcuts
- Spring Security Expressions

JSR that defines common / standard annotations

@Secured

- Springconfig.xml to enable @Secured annotations:

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

- Then on methods (class or interface):

```
public interface BankService {
  @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
  public Account readAccount(Long id);
  @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
  public Account[] findAccounts();
  @Secured("ROLE_TELLER")
  public Account post(Account account, double amount);
}
```

JSR-250 annotations

- Springconfig.xml to enable:

```
<beans:beans ...>
  ...
  <global-method-security jsr250-annotations="enabled" />
</beans:beans>
```

- Then on the class or method level:
  - @DeclareRoles(“admin”)
  - @RolesAllowed(“admin”)
  - @RunAs(“admin”)
  - @PermitAll
  - @DenyAll

AOP Pointcut

- You can specify which methods to protect:

```
<beans:beans ...>
  ...
  <global-method-security>
    <protect-pointcut expression="execution(* cs544.NormalClass.*(..))" access="ROLE_USER" />
    <protect-pointcut expression="execution(* cs544.SpecialClass.*(..))" access="ROLE_ADMIN" />
  </global-method-security>
</beans:beans>
```

Spring Security Expressions

- Springconfig.xml to enable @Secured annotations:

```
<beans:beans ...>
  ...
  <global-method-security pre-post-annotations="enabled" />
</beans:beans>
```

- Then on methods (class or interface):

```
public interface BankService {
  @PreAuthorize(“isAnonymous()")
  public Account readAccount(Long id);
  @Secured(“isAnonymous()")
  public Account[] findAccounts();
  @Secured(“hasAuthority(‘ROLE_TELLER’)")
  public Account post(Account account, double amount);
}
```

Spring Security Expressions

- Powerful Spring EL Security Expressions
  - Boolean logic and comparison operators
  - Lots of other powerful features
  - Supported with pre-post anotations
  - Supported in <http> web configuration

```
<http use-expressions="true">
  <intercept-url pattern="/admin*" access="hasRole('admin') and hasIpAddress('192.168.1.0/24')"/>
  ...
</http>
```

http://docs.spring.io/spring-security/site/docs/3.2.2.RELEASE/reference/htmlsingle/#el-access

Common Built-In Expressions

Expression	Description
hasRole([role])	Returns true if the principal has the role
hasAnyRole([role1,role2])	Returns true if the principal has any of the roles
principal	Gives direct access to the principal object
authentication	Gives direct access to the authentication object
permitAll	Always evaluates to true
denyAll	Always evaluates to false
isAnonymous()	Returns true if the principal is anonymous
isRememberMe()	Returns true if the principal is a remember-me user
isAuthenticated()	Returns true if the principal is not anonymous
isFullyAuthenticated()	Returns true if the principal is not anon or remember-me

Main Point 4

- Method security, defense in depth
- Life is found in layers

Summary

- Spring Security is a large area, we’ve covered the most important parts. Highest first

Active Learning

- How do tokens help agains CSRF?

- Why is it good to use both web and method security?