

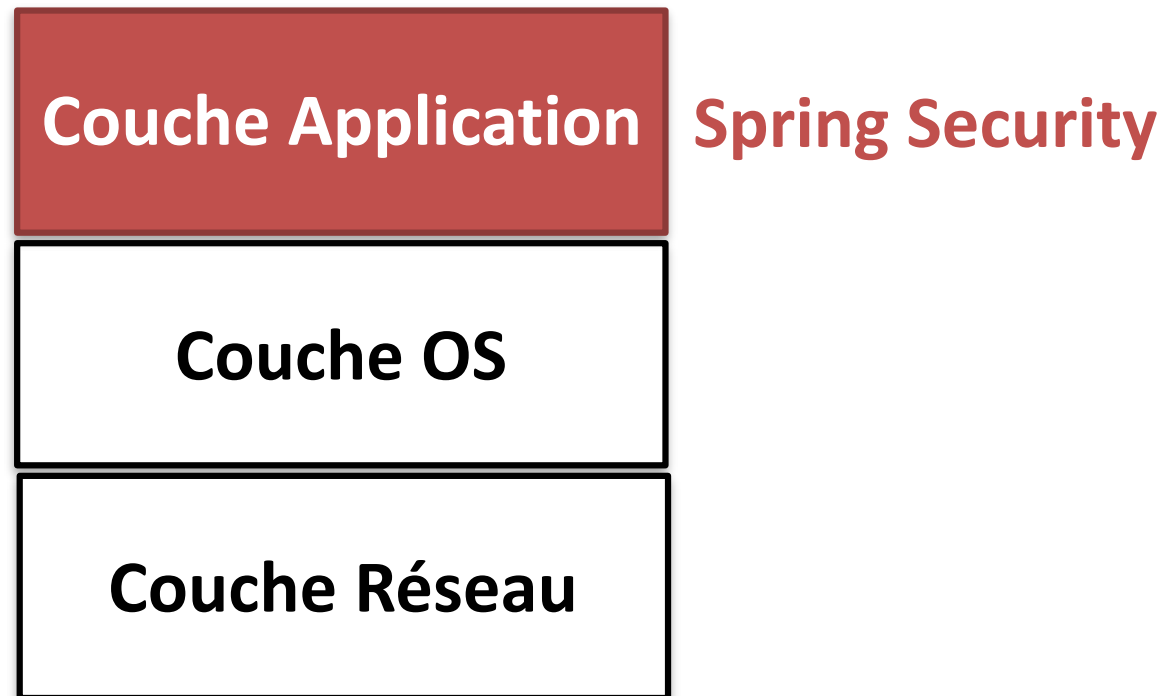
Spring Security

Introduction

Couches de sécurité

- Trois couches de sécurité

- Couche réseau
- Couche OS
- Couche Application



Authentication vs Autorisation

- Authentication

- Valide l'identité d'un utilisateur

- Autorisation

- Une fois l'utilisateur authentifié, l'accès à une ressource sécurisée est soumis à autorisation.
- Un utilisateur authentifié possède un ensemble de permissions

Concepts généraux

- Utilisateur
 - Le système
- Preuve d'identité
 - via le couple (nom utilisateur / mot de passe), via un certificat...
- Rôle
 - un groupe "logique" d'utilisateurs. Chaque groupe partage la même liste de permissions
- Ressource
 - une partie de l'application qui peut être soumise à autorisation (une URL, une méthode, un objet, ...)
- Permission
 - Droit d'accès à une ressource.
- Chiffrement
 - Masquer une information en appliquant un algorithme de transformation.

Spring Security

- Spring Security
 - Construit sur Spring Framework.
 - Fournit plusieurs modèles d'authentification : LDAP, OpenID, Base de données, ...
 - Permet de sécuriser une application à plusieurs niveaux : URL, vues, méthode, objet, ...
 - Open source.

Spring Security

- Dans quels cas utiliser Spring Security ?
 - Une application basée sur la JVM
 - Un système d'autorisation/authentification basé sur des rôles.
 - Sécuriser une application Web
 - S'intégrer avec des systèmes tiers (OpenID, LDAP, Active Directory, Base de données,...)
 - Sécuriser l'accès à un objet de l'application
 - Sécuriser son application de manière déclarative (non intrusif)
 - ...

Modules Spring Security

spring-security-core

spring-security-cas

spring-security-ldap

spring-security-config

spring-security-crypto

spring-security-remoting

spring-security-web

spring-security-acl

spring-security-taglibs

spring-security-openid

Modules Spring

- **Spring Security Core**

- Module principal sur lequel sont basés les autres modules.
- Support de l'authentification JDBC
- Support de JAAS (Java Authentication and Authorization Service)
- Encodeurs SHA et MD5
- Les packages :
 - `org.springframework.security.core`
 - `org.springframework.security.access`
 - `org.springframework.security.authentication`
 - `org.springframework.security.provisioning`

Modules Spring

- Spring Security Web
 - Filtres Web Spring Security
 - Classes liées à l'API Servlet
 - Permet de faire de l'authentification basée sur des URL
 - Package :
 - `org.springframework.security.web`

Modules Spring

- Spring Security Config
 - Support de la configuration XML
 - Support de la configuration Java
 - Package :
 - `org.springframework.security.config`

Modules Spring

- Spring Security Taglibs
 - Apporte des Tags JSP spécifiques à Spring Security

Modules Spring

- Spring Security ACL
 - Implémentation ACL (Access Control List).
 - Permet de sécuriser un objet du domaine.
 - Package :
 - `org.springframework.security.acls`

Modules Spring

- Spring Security Remoting
 - Support de l'intégration avec Spring Remoting.
 - Package principal :
 - `org.springframework.security.remoting`

Modules Spring

- Spring Security LDAP
 - Support de l'authentification LDAP
 - Package :
 - `org.springframework.security ldap`

Modules Spring

- Spring Security CAS
 - Intégration CAS.
 - Authentification SSO (Single Sign-On) à l'aide d'un serveur CAS.
 - Package :
 - `org.springframework.security.cas`

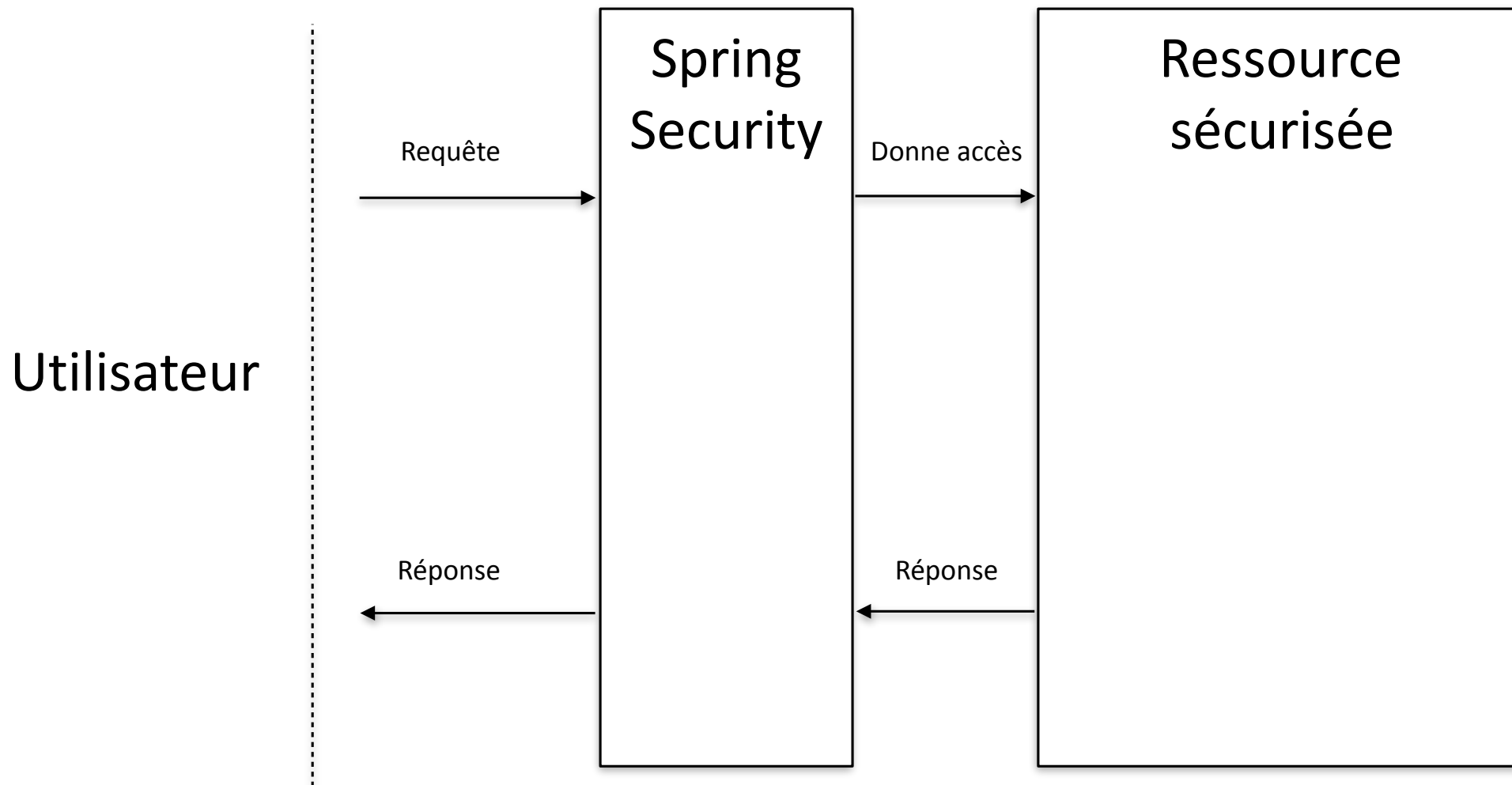
Modules Spring

- Spring Security OpenID
 - Support de l'authentification via un serveur OpenID.
 - Requiert OpenID4Java.
 - Package :
 - `org.springframework.security.openid`

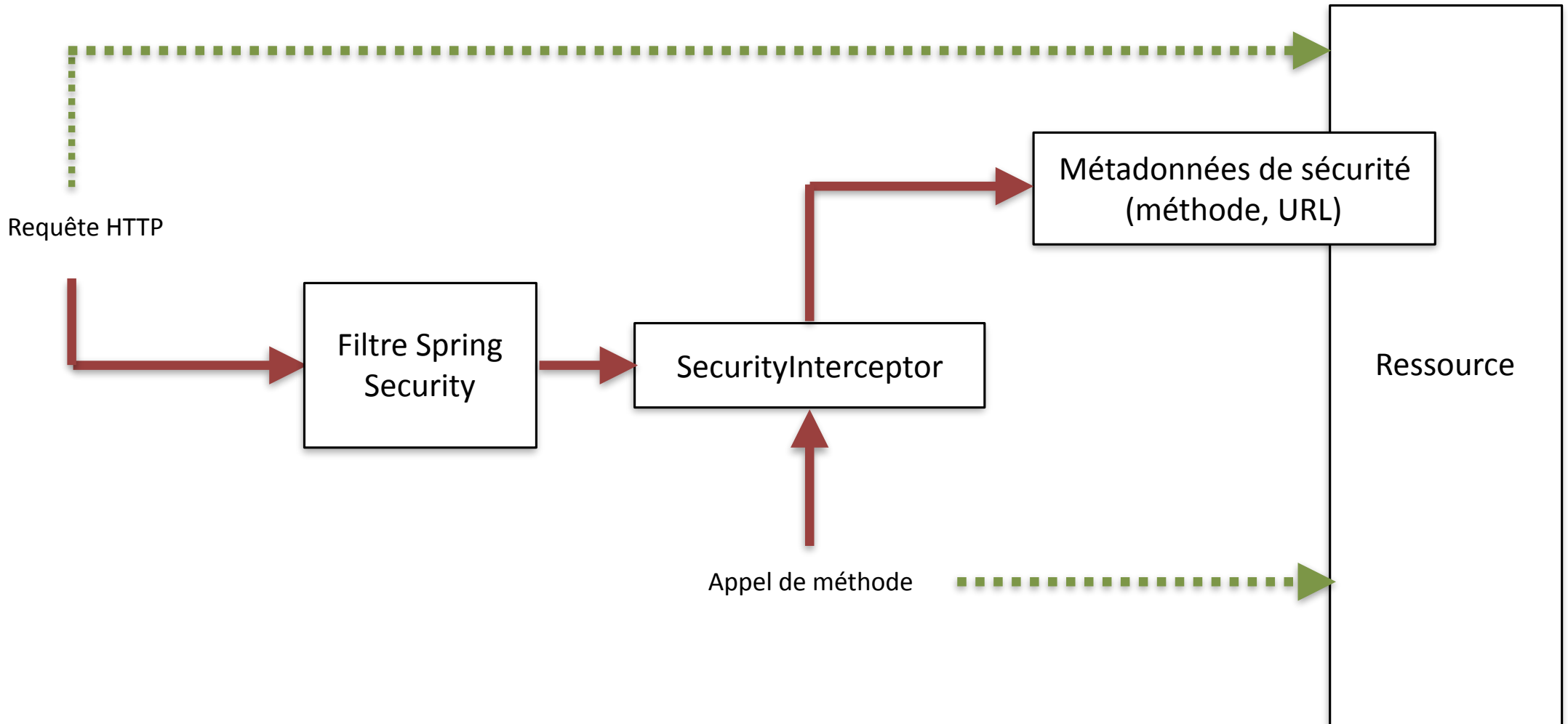
Modules Spring

- Spring Security Crypto
 - Apporte des outils de Cryptographie.
 - Exemple d'outils :
 - PasswordEncoder
 - BCryptPasswordEncoder (algorithme "bcrypt")
 - ...

Vue globale



Fonctionnement interne



Configuration XML

Configuration XML

- Nécessite le JAR **spring-security-config**.
- Permet d'ajouter des balises spécifiques Spring Security à un fichier de configuration Spring.

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:security="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.xsd  
    http://www.springframework.org/schema/security  
    http://www.springframework.org/schema/security/spring-security.xsd">  
...  
</beans>
```

WEB-INF/web.xml - Filtre Spring Security

```
<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>
```

Configuration XML

- Configuration minimale (1)

```
<beans ...>
```

```
<security:http>  
  <security:intercept-url pattern="/**" access="hasRole('USER')" />  
  <security:form-login />  
  <security:logout/>  
</security:http>
```

```
</beans>
```

- Définit que toutes les URLs de notre application sont sécurisées et requiert le rôle ROLE_USER.
- Définit que l'authentification se fera via un formulaire (nom utilisateur + mot de passe)
- Définit que l'URL de déconnexion est /logout.

Configuration XML

- Configuration minimale (2)

<beans ...>

<security:authentication-manager>

<security:authentication-provider>

<security:user-service>

<security:user authorities="ROLE_USER, ROLE_ADMIN" name="bob" password="secret" />

</security:user-service>

</security:authentication-provider>

</security:authentication-manager>

</beans>

- Définit un utilisateur avec
 - un nom utilisateur : bob
 - un mot de passe : secret

Configuration XML

- Définir une page d'authentification personnalisée

<beans ...>

```
<security:http>
  <security:intercept-url pattern="/login.jsp" access="IS_AUTHENTICATED_ANONYMOUSLY" />
  <security:intercept-url pattern="/**" access="hasRole('USER')" />
  <security:form-login login-page="/login.jsp"/>
  <security:logout/>
</security:http>
```

</beans>

- La page /login.jsp est accessible sans authentification.
- La page d'authentification est /login.jsp

Configuration XML

- default-target-url et always-use-default-target

<beans ...>

<security:http>

<security:intercept-url pattern="/login.jsp" access="IS_AUTHENTICATED_ANONYMOUSLY" />

<security:intercept-url pattern="/**" access="hasRole('USER')" />

<security:form-login login-page="/login.jsp" default-target-url="/home.jsp" always-use-default-target="true"/>

<security:logout/>

</security:http>

</beans>

- **default-target-url** permet de spécifier l'écran à afficher par défaut après authentification.
- **always-use-default-target** force l'utilisateur de la page par défaut.

Configuration XML

- Utiliser d'autres fournisseurs d'authentification

- Exemple 1

```
<security:authentication-manager>  
  <security:authentication-provider ref='myAuthenticationProvider' />  
</security:authentication-manager>
```

- Exemple 2

```
<security:authentication-manager>  
  <security:authentication-provider user-service-ref='myUserDetailsService' />  
</security:authentication-manager>  
  
<bean id="myUserDetailsService" class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">  
  <property name="dataSource" ref="dataSource" />  
</bean>
```

- Exemple 3

```
<security:authentication-manager>  
  <security:authentication-provider>  
    <jsecurity:jdbc-user-service data-source-ref="securityDataSource" />  
  </security:authentication-provider>  
</security:authentication-manager>
```

Configuration XML

- Utiliser un encodeur de mot de passe

```
<bean name="bcryptEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>

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

Configuration XML

- Différentier le protocole via l'attribut requires-channel.

```
<security:http>
  <security:intercept-url pattern="/secure/**" access="ROLE_USER" requires-channel="https"/>
  <security:intercept-url pattern="/**" access="ROLE_USER" requires-channel="any"/>
  ...
</security:http>
```

- Si l'utilisateur veut accéder en HTTP à une URL de la forme /secure/**, il est automatiquement rediriger vers l'URL HTTPS correspondante.
- Les options disponibles : http, https et any.

Configuration XML

- Détecter l'expiration de session

```
<security:http>
  ...
  <session-management invalid-session-url="/invalidSession.htm" />
  <logout delete-cookies="JSESSIONID" />
</security:http>
```

- Redirige l'utilisateur vers une page spécifique (invalidSession.htm) si la session est expirée.
- Pour éviter de confondre une expiration de session et une déconnexion, il est conseillé de supprimer le cookie de session à la déconnexion.

Configuration XML

- Contrôler le nombre de sessions utilisateur actives

- Activer un écouteur des session HTTP Spring Security

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


- Limiter le nombre de sessions actives par utilisateur. Une seconde authentification invalide la première.

```
<security:http>
  ...
  <security:session-management>
    <security:concurrency-control max-sessions="1" />
  </security:session-management>
</security:http>
```

Configuration XML

- Contrôler le nombre de sessions utilisateur actives
 - Limiter le nombre de sessions actives par utilisateur. Une seconde authentification est rejetée.

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



Configuration XML

- Sécuriser une méthode via **@Secured**

- Pour activer l'utilisation de l'annotation @Secured

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

- Il devient alors possible de sécuriser des méthodes

```
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);  
}
```

Configuration XML

- Sécuriser une méthode via la JSR-250

- Pour activer l'utilisation des annotations de la JSR-250

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

- Il devient alors possible de sécuriser des méthodes

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

Configuration XML

- Sécuriser une méthode via les annotations Pre/Post

- Pour activer l'utilisation des annotations Pre/Post (@PreAuthorize, @PostAuthorize, @PreFilter, @PostFilter)

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

- Il devient alors possible de sécuriser des méthodes

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

Configuration XML

- Sécuriser des méthodes via un point de coupe

```
<global-method-security>
```

```
  <protect-pointcut
```

```
    expression="execution(* com.mycompany.*Service.*(..))"
```

```
    access="ROLE_USER"
```

```
  />
```

```
</global-method-security>
```

Expression

Expression

- `hasRole([role])`
- `hasAnyRole([role1,role2])`
- `hasAuthority([authority])`
- `hasAnyAuthority([authority1,authority2])`
- `principal`
- `authentication`
- `permitAll`
- `denyAll`
- `isAnonymous()`
- `isRememberMe()`
- `isAuthenticated()`
- `isFullyAuthenticated()`
- `hasPermission(Object target, Object permission)`

CSRF

CSRF

- CSRF = Cross-Site Request Forgery
- Type de vulnérabilité des services d'authentification web
- Principe :
 - Faire exécuter à un utilisateur une requête HTTP falsifiée qui pointe sur une action interne au site.
- Caractéristiques
 - Implique un site qui repose sur l'authentification globale d'un utilisateur
 - Exploite cette confiance dans l'authentification pour autoriser des actions implicitement
 - Envoie des requêtes HTTP à l'insu de l'utilisateur qui est dupé de déclencher ces actions
- Prévention
 - Demander une validation supplémentaire pour les actions critiques (alourdissement du processus ?)
 - Utiliser un jeton de validité du formulaire
 - Eviter d'utiliser des requêtes GET pour effectuer des modifications du système
 - Vérifier le référent (page précédente) dans les pages sensibles
 - ...

CSRF

- Depuis Spring Security 4.0, par défaut, la protection CSRF est activée. Pour désactiver cette protection :

```
<security:http>  
    <!-- ... -->  
    <security:csrf disabled="true"/>  
</security:http>
```

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http.csrf().disable();  
}
```

CSRF

- Protéger un formulaire via la variable **_csrf**

```
<c:url var="logoutUrl" value="/logout"/>
<form action="${logoutUrl}" method="post">

    <input type="submit" value="Log out" />
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>

</form>
```

CSRF

- CSRF et requête AJAX

```
<html>
<head>
  <meta name="_csrf" content="${_csrf.token}"/>
  <!-- default header name is X-CSRF-TOKEN -->
  <meta name="_csrf_header" content="${_csrf.headerName}"/>
  <!-- ... -->
</head>
<!-- ... -->

var token = $("meta[name='_csrf']").attr("content");
var header = $("meta[name='_csrf_header']").attr("content");
$(document).ajaxSend(function(e, xhr, options) {
  xhr.setRequestHeader(header, token);
});
```

Tag JSP

Tag JSP

- Spring fournit des tags permettant d'appliquer des contraintes de sécurité dans une page JSP.
- Déclarer la librairie de tag

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
```

Tag JSP

- Tag d'authorisation

```
<sec:authorize access="hasRole('supervisor')">
```

This content will only be visible to users who have the "supervisor" authority in their list of `<tt>GrantedAuthority</tt>`s.

```
</sec:authorize>
```

```
<sec:authorize url="/admin">
```

This content will only be visible to users who are authorized to send requests to the "/admin" URL.

```
</sec:authorize>
```

Tag JSP

- Tag CSRF Input

```
<form method="post" action="/do/something">  
  <sec:csrfInput />  
  Name:<br />  
  <input type="text" name="name" />  
  ...  
</form>
```

Tag JSP

- Tag CSRF Meta

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSRF Protected JavaScript Page</title>
    <sec:csrfMetaTags />
    <script type="text/javascript" language="javascript">

      var csrfParameter = $("meta[name='_csrf_parameter']").attr("content");
      var csrfHeader = $("meta[name='_csrf_header']").attr("content");
      var csrfToken = $("meta[name='_csrf']").attr("content");

      // using XMLHttpRequest directly to send an x-www-form-urlencoded request
      var ajax = new XMLHttpRequest();
      ajax.open("POST", "http://www.example.org/do/something", true);
      ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded data");
      ajax.send(csrfParameter + "=" + csrfToken + "&name=John&...");

      // using XMLHttpRequest directly to send a non-x-www-form-urlencoded request
      var ajax = new XMLHttpRequest();
      ajax.open("POST", "http://www.example.org/do/something", true);
      ajax.setRequestHeader(csrfHeader, csrfToken);
      ajax.send("...");
```


Integration Spring MVC

Integration Spring MVC

- **Avant Spring 4.0**, l'annotation **@EnableWebMvcSecurity** permet d'activer l'intégration Spring Security dans un projet Spring MVC.
- **Depuis Spring 4.0**, l'annotation **@EnableWebSecurity** suffit. Si Spring MVC est le classpath l'intégration est activée.

Integration Spring MVC

- Récupérer des informations d'authentification depuis un contrôleur

```
@RequestMapping("/messages/inbox")
public ModelAndView findMessagesForUser() {

    Authentication authentication =
        SecurityContextHolder.getContext().getAuthentication();

    CustomUser custom = (CustomUser) authentication == null ? null :
authentication.getPrincipal();

    // .. find messags for this user and return them ...
}

@RequestMapping("/messages/inbox")
public ModelAndView findMessagesForUser(@AuthenticationPrincipal CustomUser customUser) {

    // .. find messags for this user and return them ...
}
```

Services Core

Gestionnaire d'authentification

- L'**AuthenticationManager** est une interface représentant le gestionnaire d'authentification.

```
public interface AuthenticationManager {  
    Authentication authenticate(Authentication authentication)  
        throws AuthenticationException;  
}
```

- L'implémentation par défaut s'appelle "**ProviderManager**" qui délègue le traitement de l'authentification à une liste de composants qui implémentent l'interface "**AuthProvider**".

```
public interface AuthenticationProvider {  
    Authentication authenticate(Authentication authentication)  
        throws AuthenticationException;  
  
    boolean supports(Class<?> authentication);  
}
```

Gestionnaire d'authentification

- Chaque composant AuthenticationProvider peut soit retourner une exception ou retourner un objet Authentication. Lors de l'utilisation de la balise authentication-manager, une instance du ProviderManager est créée et maintenu en interne.

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

- Il est possible de déclarer soit même le ProviderManager

```
<bean id="authenticationManager"  
  class="org.springframework.security.authentication.ProviderManager">  
  <constructor-arg>  
    <list>  
      <ref local="daoAuthenticationProvider"/>  
      <ref local="anonymousAuthenticationProvider"/>  
      <ref local="ldapAuthenticationProvider"/>  
    </list>  
  </constructor-arg>  
</bean>
```

Services Core

- Authentification en mémoire

```
<user-service id="userDetailsService" properties="users.properties">  
  <user name="jimi" password="jimispASSWORD" authorities="ROLE_USER, ROLE_ADMIN" />  
  <user name="bob" password="bobspASSWORD" authorities="ROLE_USER" />  
</user-service>
```

- Format des lignes du fichier users.properties
 - username=password,grantedAuthority[,grantedAuthority][,enabled|disabled]
 - Exemples :
 - jimi=jimispASSWORD,ROLE_USER,ROLE_ADMIN,enabled
 - bob=bobspASSWORD,ROLE_USER,enabled

Services Core

- Authentification via une base de données

```
<bean id="dataSource"  
class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>  
    <property name="url" value="jdbc:hsqldb:hsqldb://localhost:9001"/>  
    <property name="username" value="sa"/>  
    <property name="password" value=""/>  
</bean>  
  
<bean id="userDetailsService"  
    class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">  
    <property name="dataSource" ref="dataSource"/>  
</bean>
```


Services Core

- Authentication via une base de données (le schéma)

```
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);
```

Services Core

- Authentication via une base de données (le schéma) - si les groupes sont activés

```
create table groups (  
    id bigint generated by default as identity(start with 0) primary key,  
    group_name varchar_ignorecase(50) not null  
);  
  
create table group_authorities (  
    group_id bigint not null,  
    authority varchar(50) not null,  
    constraint fk_group_authorities_group foreign key(group_id) references groups(id)  
);  
  
create table group_members (  
    id bigint generated by default as identity(start with 0) primary key,  
    username varchar(50) not null,  
    group_id bigint not null,  
    constraint fk_group_members_group foreign key(group_id) references groups(id)  
);
```

Configuration Java

Configuration Java

- La configuration Java a été ajoutée dans Spring 3.1. Depuis Spring 3.2, il est possible de se passer entièrement de la configuration XML.
- La configuration Java peut s'activer avec une des annotations suivantes :
@EnableWebSecurity, @EnableGlobalMethodSecurity ou @EnableGlobalAuthentication

@EnableWebSecurity

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Autowired  
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
        auth.inMemoryAuthentication().withUser("user").password("password").roles("USER")  
    };  
}
```

Configuration Java

- Par défaut, cette configuration fournit les services suivants :
 - Requiert une authentification pour accéder aux ressources sécurisées (toutes dans ce cas).
 - Génère le formulaire d'authentification.
 - Permet la déconnexion
 - Les attaques CSRF sont gérées
 - Une intégration avec la Servlet API est fournie (`HttpServletRequest.isUserInRole(String)`, `HttpServletRequest.logout()`, ...)

Configuration Java

- Se passer de web.xml

```
public class SecurityWebApplicationInitializer
    extends AbstractSecurityWebApplicationInitializer {

    public SecurityWebApplicationInitializer() {
        super(SecurityConfig.class);
    }
}
```

- Enregistre automatiquement un filtre springSecurityFilterChain pour toutes les URL
- Charge la configuration de la classe SecurityConfig

Configuration Java

- Une API permettant de décrire des informations similaires à la configuration XML

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
        .anyRequest().authenticated()  
        .and()  
        .formLogin()  
        .and()  
        .httpBasic();  
}
```

```
<http>  
    <intercept-url pattern="/**" access="authenticated"/>  
    <form-login />  
    <http-basic />  
</http>
```

Configuration Java

- Formulaire de connexion

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
        .anyRequest().authenticated()  
        .and()  
        .formLogin()  
        .loginPage("/login")  
        .permitAll();  
}
```


Configuration Java

- Sécurité des requêtes

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
        .antMatchers("/resources/**", "/signup", "/about").permitAll()  
        .antMatchers("/admin/**").hasRole("ADMIN")  
        .antMatchers("/db/**").access("hasRole('ADMIN') and hasRole('DBA')")  
        .anyRequest().authenticated()  
        .and()  
        // ...  
        .formLogin();  
}
```

Configuration Java

- Gérer la déconnexion

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .logout()  
        .logoutUrl("/my/logout")  
        .logoutSuccessUrl("/my/index")  
        .logoutSuccessHandler(logoutSuccessHandler)  
        .invalidateHttpSession(true)  
        .addLogoutHandler(logoutHandler)  
        .deleteCookies(cookieNamesToClear)  
        .and()  
        ...  
}
```

Configuration Java

- Authentication - En mémoire

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .inMemoryAuthentication()
        .withUser("user").password("password").roles("USER").and()
        .withUser("admin").password("password").roles("USER", "ADMIN");
}
```

Configuration Java

- Authentication - JDBC

```
@Autowired  
private DataSource dataSource;
```

```
@Autowired  
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
    auth  
        .jdbcAuthentication()  
        .dataSource(dataSource)  
        .withDefaultSchema()  
        .withUser("user").password("password").roles("USER").and()  
        .withUser("admin").password("password").roles("USER", "ADMIN");  
}
```

Configuration Java

- Créer son fournisseur d'authentification

```
@Bean
public SpringAuthenticationProvider springAuthenticationProvider() {
    return new SpringAuthenticationProvider();
}
```

Configuration Java

- Sécurité des méthodes

```
@EnableGlobalMethodSecurity(securedEnabled = true)
public class MethodSecurityConfig {
    // ...
}
```

```
@EnableGlobalMethodSecurity(jsr250Enabled = true)
public class MethodSecurityConfig {
    // ...
}
```

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MethodSecurityConfig {
    // ...
}
```