Migrating from Spring Security 3.x to 4.x (XML Configuration)

Rob Winch
@rob_winch (https://twitter.com/rob_winch)

Table of Contents

This guide is intended to help users migrate from Spring Security 3.x to Spring Security 4.x when using **XML based configuration**.

If you are looking to migrate from Spring Security 3.x to Spring Security 4.x when using Java Based configuration, click here

1. Introduction

As exploits against applications evolve, so must Spring Security. As a major release version, the Spring Security team took the opportunity to make some non-passive changes which focus on:

- Ensuring Spring Security is more secure by default (https://www.owasp.org/index.php/Establish_secure_defaults)
- Minimizing Information Leakage (https://www.owasp.org/index.php/Information_Leakage)
- Removing deprecated APIs

A complete listing of non-passive changes between 3.x and 4.x can be found in JIRA
(https://jira.spring.io/issues/?
jql=project%20%3D%20SEC%20AND%20fixVersion%20in%20(4.0.0%2C%204.0.0.M1%2C%204.0.0.M2%2C%204.0.0.RC1%2C%204.0.0.RC2)%20AND%20labels%2(

2. Sample Migration

A sample illustrating all of the changes in a migration can be found on github (https://github.com/spring-projects/spring-security-migrate-3-to-4/). The sample demonstrates migrating spring-security-3-xml (https://github.com/spring-projects/spring-security-migrate-3-to-4/tree/master/xml/spring-security-3-xml) to Spring Security 4. The completed migration can be found in spring-security-4-xml (https://github.com/spring-projects/spring-security-migrate-3-to-4/tree/master/xml/spring-security-4-xml)

You can find a diff of the changes on github (https://github.com/spring-projects/spring-security-migrate-3-to-4/compare/xml?expand=1).

3. Updating to Spring 4.1.x

Spring Security 4 now requires Spring 4. Conveniently, Spring Security 3.2.x works with Spring 3.2.x and Spring 4. This means your first step is to update to Spring 4.1.x.

For detailed instructions refer to:

- Migrating to Spring Framework 4.0 (https://github.com/spring-projects/spring-framework/wiki/Migrating-from-earlier-versions-of-the-spring-framework#migrating-to-spring-framework-40)
- Migrating to Spring Framework 4.1 (https://github.com/spring-projects/spring-framework/wiki/Migrating-from-earlier-versions-of-the-spring-framework#migrating-to-spring-framework-41)

4. Deprecations

A number of deprecations were removed in Spring Security 4 to clean up clutter. The following section describes how to migrate each deprecation.

If you are using the XML Namespace configuration, there are many instances where you will be shielded from deprecation. If you (or a non-spring library you use) do NOT use an API directly, then you will NOT be impacted. This means typically a quick search in your workspace should allow you to find all the deprecations.

4.1. Related Links

For thoroughness we have include the related links in the table below.

JIRA	Commits
SEC-2781 (https://jira.spring.io/browse/SEC-2781)	6e204ff (https://github.com/spring-projects/spring-security/commit/6e204fff72b80196a83245cbc3bd0cd401feda00)

4.2. spring-security-acl

This section describes all of the deprecated APIs within the spring-security-acl module. If you are not using the spring-security-acl module, you can safely skip to spring-security-cas.

4.2.1. AclImpl

AclImpl had a deprecated constructor removed. Specifically, the constructor that defaults the `PermissionGrantingStrategy` was removed:

```
JAVA
@Deprecated
public AclImpl(ObjectIdentity objectIdentity, Serializable id,
                        AclAuthorizationStrategy aclAuthorizationStrategy,
                        AuditLogger auditLogger, Acl parentAcl,
                ...
                        List<Sid> loadedSids, boolean entriesInheriting, Sid owner) {
}
```

This means that an AclImpl was being created with this constructor:

```
JAVA
new AclImpl(objectIdentity, id, aclAuthorizationStrategy, auditLogger,
                                                                parentAcl, loadedSids, entriesInheriting, owner);
```

it needs to be updated to pass in the `PermissionGrantingStrategy` instead of the `AuditLogger`

```
JAVA
PermissionGrantingStrategy permissionGrantingStrategy =
        new DefaultPermissionGrantingStrategy(auditLogger);
new AclImpl(objectIdentity, id, aclAuthorizationStrategy, permissionGrantingStrategy,
                                            parentAcl, loadedSids, entriesInheriting, owner);
```

4.2.2. EhCacheBasedAclCache

`EhCacheBasedAclCache` had a deprecated constructor removed. Specifically, the constructor that defaults the `PermissionGrantingStrategy` was removed:

```
JAVA
@Deprecated
public EhCacheBasedAclCache(Ehcache cache) {
        ...
}
```

This means that an `EhCacheBasedAclCache` was being created with this constructor:

```
JAVA
new EhCacheBasedAclCache(ehCache);
```

it needs to be updated to pass in the `PermissionGrantingStrategy` and `AclAuthorizationStrategy` too:

```java
PermissionGrantingStrategy permissionGrantingStrategy =
                                new DefaultPermissionGrantingStrategy(auditLogger);
AclAuthorizationStrategy aclAuthorizationStrategy =
                new AclAuthorizationStrategyImpl(new SimpleGrantedAuthority("ROLE_ACL_ADMIN"));
new EhCacheBasedAclCache(ehCache, permissionGrantingStrategy, aclAuthorizationStrategy);
```

and

```xml
<bean id="aclCache" class="org.springframework.security.acls.domain.EhCacheBasedAclCache">
<constructor-arg>
        <bean class="org.springframework.cache.ehcache.EhCacheFactoryBean">
        <property name="cacheManager">
                <bean class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
        </property>
        <property name="cacheName" value="aclCache"/>
        </bean>
</constructor-arg>
</bean>
```

needs to be update to

```xml
<bean id="aclCache" class="org.springframework.security.acls.domain.EhCacheBasedAclCache">
<constructor-arg>
        <bean class="org.springframework.cache.ehcache.EhCacheFactoryBean">
        <property name="cacheManager">
                <bean class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
        </property>
        <property name="cacheName" value="aclCache"/>
        </bean>
</constructor-arg>
<constructor-arg>
        <bean class="org.springframework.security.acls.domain.DefaultPermissionGrantingStrategy">
                <constructor-arg>
                        <bean class="org.springframework.security.acls.domain.ConsoleAuditLogger"/>
                </constructor-arg>
        </bean>
</constructor-arg>
<constructor-arg>
        <bean class="org.springframework.security.acls.domain.AclAuthorizationStrategyImpl">
                <constructor-arg>
                        <list>
                                <bean class="org.springframework.security.core.authority.SimpleGrantedAuthority">
                                        <constructor-arg value="ROLE_ACL_ADMIN"/>
                                </bean>
                        </list>
                </constructor-arg>
        </bean>
</constructor-arg>
</bean>
```

4.3. spring-security-cas

This section describes all of the deprecated APIs within the spring-security-cas module. If you are not using the spring-security-cas module, you can safely skip to spring-security-config.

4.3.1. ServiceAuthenticationDetailsSource

`ServiceAuthenticationDetailsSource` removed the deprecated construtors that defaulted the `ServiceProperties`.

```java
@Deprecated
public ServiceAuthenticationDetailsSource() {
        ...
}

@Deprecated
public ServiceAuthenticationDetailsSource(final String artifactParameterName) {
        ...
}
```

This means that an `ServiceAuthenticationDetailsSource` was being created with these constructors:

```java
new ServiceAuthenticationDetailsSource();

new ServiceAuthenticationDetailsSource(artifactId);
```

it needs to be updated to pass in the `ServiceProperties` as shown below:

```java
new ServiceAuthenticationDetailsSource(serviceProperties);

new ServiceAuthenticationDetailsSource(serviceProperties, artifactId);
```

and

```xml
<b:bean class="org.springframework.security.cas.web.authentication.ServiceAuthenticationDetailsSource"/>

<b:bean class="org.springframework.security.cas.web.authentication.ServiceAuthenticationDetailsSource">
        <b:constructor-arg value="TICKET"/>
</b:bean>
```

needs to be updated to

```xml
<b:bean class="org.springframework.security.cas.web.authentication.ServiceAuthenticationDetailsSource">
        <b:constructor-arg ref="serviceProperties"/>
</b:bean>

<b:bean class="org.springframework.security.cas.web.authentication.ServiceAuthenticationDetailsSource">
        <b:constructor-arg ref="serviceProperties"/>
        <b:constructor-arg value="TICKET"/>
</b:bean>
```

4.4. spring-security-config

This section describes all of the deprecated APIs within the spring-security-config module. If you are not using the spring-security-config module or have already completed this task, you can safely skip to spring-security-core.

4.4.1. filter-invocation-definition-source

The XML element `filter-invocation-definition-source` was removed in favor of filter-security-metadata-source (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-filter-security-metadata-source). This means if you have something like this:

```xml
<filter-invocation-definition-source ...>
        ...
</filter-invocation-definition-source>
```

it needs to be replaced with:

```xml
<filter-security-metadata-source ...>
        ...
</filter-security-metadata-source>
```

4.4.2. http@access-denied-page

The XML attribute `http@access-denied-page` was removed in favor of access-denied-handler@error-page (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-access-denied-handler-error-page). This means if you have something like this:

```xml
<http ... access-denied-page="/denied">
        ...
</http>
```

it needs to be replaced with:

```xml
<http ...>
        <access-denied-handler error-page="/denied"/>
</http>
```

4.4.3. http@path-type

The XML attribute `http@path-type` was removed in favor of http@request-matcher (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-http-request-matcher). This means if you have something like this:

```xml
<http ... path-type="regex">
        ...
</http>
```

it needs to be replaced with:

```xml
<http ... request-matcher="regex">
        ...
</http>
```

4.4.4. filter-chain-map@path-type

The XML attribute `filter-chain-map@path-type` was removed in favor of filter-chain-map@request-matcher (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-filter-chain-map-request-matcher). This means if you have something like this:

```xml
<filter-chain-map ... path-type="regex">
        ...
</filter-chain-map>
```

it needs to be replaced with:

```xml
<filter-chain-map ... request-matcher="regex">
        ...
</filter-chain-map>
```

4.4.5. filter-security-metadata-source@path-type

The XML attribute `filter-security-metadata-source@path-type` was removed in favor of filter-security-metadata-source@request-matcher (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-filter-security-metadata-source-request-matcher). This means if you have something like this:

```xml
<filter-security-metadata-source ... path-type="regex">
        ...
</filter-security-metadata-source>
```

it needs to be replaced with:

```xml
<filter-security-metadata-source ... request-matcher="regex">
        ...
</filter-security-metadata-source>
```

4.5. spring-security-core

This section describes all of the deprecated APIs within the spring-security-core module. If you are not using the spring-security-core module or have already completed this task, you can safely skip to spring-security-openid.

4.5.1. SecurityConfig

`SecurityConfig.createSingleAttributeList(String)` was removed in favor of using `SecurityConfig.createList(String…)`. This means if you have something like this:

```java
List<ConfigAttribute> attrs =
                SecurityConfig.createSingleAttributeList("ROLE_USER");
```

needs to be replaced with:

```java
List<ConfigAttribute> attrs =
                SecurityConfig.createList("ROLE_USER");
```

4.5.2. UserDetailsServiceWrapper

`UserDetailsServiceWrapper` was deprecated in favor of using `RoleHierarchyAuthoritiesMapper`. For example, if you have something like this:

```xml
<authentication-manager>
        <authentication-provider user-service-ref="userDetailsServiceWrapper"/>
</authentication-manager>

<b:bean id="userDetailsServiceWrapper"
class="org.springframework.security.access.hierarchicalroles.UserDetailsServiceWrapper">
        <b:property name="userDetailsService" ref="userDetailsService"/>
        <b:property name="roleHierarchy" ref="roleHierarchy"/>
</b:bean>

<b:bean id="roleHierarchy" class="org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl">
        <b:property name="hierarchy">
                <b:value>
                        ROLE_ADMIN > ROLE_USER
                </b:value>
        </b:property>
</b:bean>
```

then it needs to be migrated with something like this:

```xml
<authentication-manager>
        <authentication-provider ref="authenticationProvider"/>
</authentication-manager>

<b:bean id="authenticationProvider" class="org.springframework.security.authentication.dao.DaoAuthenticationProvider">
        <b:property name="userDetailsService" ref="userDetailsService"/>
        <b:property name="authoritiesMapper" ref="authoritiesMapper"/>
</b:bean>

<b:bean id="authoritiesMapper"
class="org.springframework.security.access.hierarchicalroles.RoleHierarchyAuthoritiesMapper">
        <b:constructor-arg ref="roleHierarchy"/>
</b:bean>

<b:bean id="roleHierarchy" class="org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl">
        <b:property name="hierarchy">
                <b:value>
                        ROLE_ADMIN > ROLE_USER
                </b:value>
        </b:property>
</b:bean>
```

4.5.3. UserDetailsWrapper

`UserDetailsWrapper` was deprecated in favor of using `RoleHierarchyAuthoritiesMapper`. Typically users would not use the `UserDetailsWrapper` directly. However, if they are they can use `RoleHierarchyAuthoritiesMapper` For example, if the following code is present:

```java
UserDetailsWrapper authenticate = new UserDetailsWrapper(userDetails, roleHiearchy);
```

then it needs to be replaced by:

```java
Collection<GrantedAuthority> allAuthorities =
                roleHiearchy.getReachableGrantedAuthorities(userDetails.getAuthorities());
UserDetails authenticate =
                new User(userDetails.getUsername(), userDetails.getPassword(), allAuthorities);
```

4.5.4. AbstractAccessDecisionManager

The default constructor for `AbstractAccessDecisionManager` has been deprecated along with the `setDecisionVoters` method. Naturally, this impacts the subclasses `AffirmativeBased`, `ConsensusBased`, and `UnanimousBased`. For example, this means that if you are using the following:

```java
AffirmativeBased adm = new AffirmativeBased();
adm.setDecisionVoters(voters);
```

it needs to be migrated to:

```java
AffirmativeBased adm = new AffirmativeBased(voters);
```

This type of migration also applies to XML based configuration. For example, if you are using the following:

```xml
<b:bean class="org.springframework.security.access.vote.UnanimousBased">
        <b:property name="decisionVoters" ref="voters"/>
</b:bean>
```

then it needs to be migrated to:

```xml
<b:bean class="org.springframework.security.access.vote.UnanimousBased">
        <b:constructor-arg ref="voters"/>
</b:bean>
```

4.5.5. AuthenticationException

The constructor that accepts extraInformation within `AuthenticationException` was removed to prevent accidental leaking of the `UserDetails`. Specifically, the following we removed.

```java
public AccountExpiredException(String msg, Object extraInformation) {
...
}
```

This impacts the subclasses `AccountStatusException`, `AccountExpiredException`, `BadCredentialsException`, `CredentialsExpiredException`, `DisabledException`, `LockedException`, and `UsernameNotFoundException`. If use are using any of these constructors, simply remove the additional argument. For example, the following is changed from:

```java
new LockedException("Message", userDetails);
```

to:

```java
new LockedException("Message");
```

4.5.6. AnonymousAuthenticationProvider

`AnonymousAuthenticationProvider` default constructor and `setKey` method was deprecated in favor of using constructor injection. For example, if you have the following:

```java
AnonymousAuthenticationProvider provider = new AnonymousAuthenticationProvider();
provider.setKey(key);
```

it should be changed to:

```java
AnonymousAuthenticationProvider provider = new AnonymousAuthenticationProvider(key);
```

4.5.7. AuthenticationDetailsSourceImpl

`AuthenticationDetailsSourceImpl` was deprecated in favor of writing a custom `AuthenticationDetailsSource`. For example, if you have the following:

```java
AuthenticationDetailsSourceImpl source = new AuthenticationDetailsSourceImpl();
source.setClazz(CustomWebAuthenticationDetails.class);
```

You should implement `AuthenticationDetailsSource` directly to return `CustomSource`:

```java
public class CustomWebAuthenticationDetailsSource implements AuthenticationDetailsSource<HttpServletRequest,
WebAuthenticationDetails> {

        public WebAuthenticationDetails buildDetails(HttpServletRequest context) {
                return new CustomWebAuthenticationDetails(context);
        }
}
```

4.5.8. ProviderManager

`ProviderManager` has removed the deprecated default constructor and the correspdonding setter methods in favor of using constructor injection. It has also removed the clearExtraInformation property since the `AuthenticationException` had the extra information property removed.

For example, if you have something like the following:

```
ProviderManager provider = new ProviderManager();
provider.setParent(parent);
provider.setProviders(providers);
provider.setClearExtraInformation(true);
```

JAVA

then it should be changed to:

```
ProviderManager provider = new ProviderManager(providers, parent);
```

JAVA

The `clearExtraInformation` property was removed since the `AuthenticationException` had the extra information property removed. So there is no replacement for this.

and

```
<b:bean class="org.springframework.security.authentication.ProviderManager">
        <b:property name="parent" ref="parent"/>
        <b:property name="providers">
                <b:list>
                        <b:ref bean="authenticationProvider"/>
                </b:list>
        </b:property>
        <b:property name="clearExtraInformation" value="true"/>
</b:bean>
```

XML

should be changed to

```
<b:bean class="org.springframework.security.authentication.ProviderManager">
        <b:constructor-arg>
                <b:list>
                        <b:ref bean="authenticationProvider"/>
                </b:list>
        </b:constructor-arg>
        <b:constructor-arg ref="parent"/>
</b:bean>
```

XML

4.5.9. RememberMeAuthenticationProvider

`RememberMeAuthenticationProvider` had the default constructor and the `setKey` method removed in favor of constructor injection. For example:

```
RememberMeAuthenticationProvider provider = new RememberMeAuthenticationProvider();
provider.setKey(key);
```

JAVA

should be migrated to:

```
RememberMeAuthenticationProvider provider = new RememberMeAuthenticationProvider(key);
```

JAVA

and

```
<b:bean class="org.springframework.security.authentication.RememberMeAuthenticationProvider">
        <b:property name="key" value="key"/>
</b:bean>
```

XML

should be migrated to

```
<b:bean class="org.springframework.security.authentication.RememberMeAuthenticationProvider">
        <b:constructor-arg value="key"/>
</b:bean>
```

XML

4.5.10. GrantedAuthorityImpl

`GrantedAuthorityImpl` was removed in favor of `SimpleGrantedAuthority` or implementing your own. For example:

```java
new GrantedAuthorityImpl(role);
```

should be replaced with

```java
new SimpleGrantedAuthority(role);
```

4.5.11. InMemoryDaoImpl

`InMemoryDaoImpl` was replaced in favor of `InMemoryUserDetailsManager`

For example the following:

```java
InMemoryDaoImpl uds = new InMemoryDaoImpl();
uds.setUserProperties(properties);
```

should be replaced with

```java
InMemoryUserDetailsManager uds = new InMemoryUserDetailsManager(properties);
```

and

```xml
<b:bean class="org.springframework.security.core.userdetails.memory.InMemoryDaoImpl">
        <b:property name="userProperties">
                <b:value>
                user=password,ROLE_USER
                </b:value>
        </b:property>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.provisioning.InMemoryUserDetailsManager">
        <b:constructor-arg>
                <b:value>
                user=password,ROLE_USER
                </b:value>
        </b:constructor-arg>
</b:bean>
```

4.5.12. spring-security-openid

This section describes all of the deprecated APIs within the spring-security-openid module. If you are not using the spring-security-openid module or have already completed this task, you can safely skip to spring-security-taglibs.

4.5.13. OpenID4JavaConsumer

The `OpenID4JavaConsumer` constructors that accept `List<OpenIDAttribute>` have been removed in favor of using an `AxFetchListFactory`. For example:

```java
new OpenID4JavaConsumer(attributes);
```

should be replaced with:

```java
Map<String, List<OpenIDAttribute>> regexMap = new HashMap<String,List<OpenIDAttribute>>();
regexMap.put(".*", attributes);
RegexBasedAxFetchListFactory factory = new RegexBasedAxFetchListFactory(regexMap);
new OpenID4JavaConsumer(factory);
```

and

```xml
<b:bean class="org.springframework.security.openid.OpenID4JavaConsumer">
        <b:constructor-arg>
                <b:list>
                        <b:bean class="org.springframework.security.openid.OpenIDAttribute">
                                <b:constructor-arg value="email"/>
                                <b:constructor-arg value="http://axschema.org/contact/email"/>
                        </b:bean>
                </b:list>
        </b:constructor-arg>
</b:bean>
```

should be replaced with:

```xml
<b:bean class="org.springframework.security.openid.OpenID4JavaConsumer">
        <b:constructor-arg>
                <b:bean class="org.springframework.security.openid.RegexBasedAxFetchListFactory">
                        <b:constructor-arg>
                                <b:map>
                                        <b:entry key=".*">
                                                <b:list>
                                                        <b:bean
class="org.springframework.security.openid.OpenIDAttribute">
                                                                <b:constructor-arg value="email"/>
                                                                <b:constructor-arg
value="http://axschema.org/contact/email"/>
                                                        </b:bean>
                                                </b:list>
                                        </b:entry>
                                </b:map>
                        </b:constructor-arg>
                </b:bean>
        </b:constructor-arg>
</b:bean>
```

4.6. spring-security-taglibs

This section describes all of the deprecated APIs within the spring-security-taglibs module. If you are not using the spring-security-taglibs module or have already completed this task, you can safely skip to spring-security-web.

Spring Security's authorize JSP tag deprecated the properties `ifAllGranted`, `ifAnyGranted`, and `ifNotGranted` in favor of using expressions.

For example:

```xml
<sec:authorize ifAllGranted="ROLE_ADMIN,ROLE_USER">
        <p>Must have ROLE_ADMIN and ROLE_USER</p>
</sec:authorize>
<sec:authorize ifAnyGranted="ROLE_ADMIN,ROLE_USER">
        <p>Must have ROLE_ADMIN or ROLE_USER</p>
</sec:authorize>
<sec:authorize ifNotGranted="ROLE_ADMIN,ROLE_USER">
        <p>Must not have ROLE_ADMIN or ROLE_USER</p>
</sec:authorize>
```

can be replaced with:

```xml
<sec:authorize access="hasRole('ROLE_ADMIN') and hasRole('ROLE_USER')">
        <p>Must have ROLE_ADMIN and ROLE_USER</p>
</sec:authorize>
<sec:authorize access="hasAnyRole('ROLE_ADMIN','ROLE_USER')">
        <p>Must have ROLE_ADMIN or ROLE_USER</p>
</sec:authorize>
<sec:authorize access="!hasAnyRole('ROLE_ADMIN','ROLE_USER')">
        <p>Must not have ROLE_ADMIN or ROLE_USER</p>
</sec:authorize>
```

4.7. spring-security-web

This section describes all of the deprecated APIs within the spring-security-taglibs module. If you are not using the spring-security-taglibs module or have already completed this task, you can safely skip to [m3to4-xml -defaults].

4.7.1. FilterChainProxy

`FilterChainProxy` removed the `setFilterChainMap` method in favor of constructor injection. For example, if you have the following:

```
JAVA
FilterChainProxy filter = new FilterChainProxy();
filter.setFilterChainMap(filterChainMap);
```

it should be replaced with:

```
JAVA
FilterChainProxy filter = new FilterChainProxy(securityFilterChains);
```

`FilterChainProxy` also removed `getFilterChainMap` in favor of using `getFilterChains` for example:

```
JAVA
FilterChainProxy securityFilterChain = ...
Map<RequestMatcher,List<Filter>> mappings = securityFilterChain.getFilterChainMap();
for(Map.Entry<RequestMatcher, List<Filter>> entry : mappings.entrySet()) {
        RequestMatcher matcher = entry.getKey();
        boolean matches = matcher.matches(request);
        List<Filter> filters = entry.getValue();
}
```

should be replaced with

```
JAVA
FilterChainProxy securityFilterChain = ...
List<SecurityFilterChain> mappings = securityFilterChain.getFilterChains();
for(SecurityFilterChain entry : mappings) {
        boolean matches = entry.matches(request);
        List<Filter> filters = entry.getFilters();
}
```

and

```
XML
<b:bean class="org.springframework.security.web.FilterChainProxy">
        <b:property name="filterChainMap">
                <b:map>
                        <b:entry key="#{T(org.springframework.security.web.util.matcher.AnyRequestMatcher).INSTANCE}">
                                <b:ref bean="mockFilter"/>
                        </b:entry>
                </b:map>
        </b:property>
</b:bean>
```

should be replaced with

```
XML
<b:bean class="org.springframework.security.web.FilterChainProxy">
        <b:constructor-arg>
                <b:bean class="org.springframework.security.web.DefaultSecurityFilterChain">
                        <b:constructor-arg value="#
{T(org.springframework.security.web.util.matcher.AnyRequestMatcher).INSTANCE}"/>
                        <b:constructor-arg ref="mockFilter"/>
                </b:bean>
        </b:constructor-arg>
</b:bean>
```

4.7.2. ExceptionTranslationFilter

The default constructor for `ExceptionTranslationFilter` and the `setAuthenticationEntryPoint` method was removed in favor of using constructor injection.

```
JAVA
ExceptionTranslationFilter filter = new ExceptionTranslationFilter();
filter.setAuthenticationEntryPoint(entryPoint);
filter.setRequestCache(requestCache);
```

can be replaced with

```
JAVA
ExceptionTranslationFilter filter = new ExceptionTranslationFilter(entryPoint, requestCache);
```

and

```xml
<b:bean class="org.springframework.security.web.access.ExceptionTranslationFilter">
        <b:property name="authenticationEntryPoint" ref="entryPoint"/>
        <b:property name="requestCache" ref="requestCache"/>
</b:bean>
```

can be replaced with

```xml
<b:bean class="org.springframework.security.web.access.ExceptionTranslationFilter">
        <b:constructor-arg ref="entryPoint"/>
        <b:constructor-arg ref="requestCache"/>
</b:bean>
```

4.7.3. AbstractAuthenticationProcessingFilter

`AbstractAuthenticationProcessingFilter` had its `successfulAuthentication(HttpServletRequest,HttpServletResponse,Authentication)` method removed. So if your application overrides the following method:

```java
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                        Authentication authResult) throws IOException, ServletException {
}
```

it should be replaced with:

```java
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                        FilterChain chain, Authentication authResult) throws IOException,
ServletException {
}
```

4.7.4. AnonymousAuthenticationFilter

`AnonymousAuthenticationFilter` had the default constructor and the `setKey` and `setPrincipal` methods removed in favor of constructor injection. For example:

```java
AnonymousAuthenticationFilter filter = new AnonymousAuthenticationFilter();
filter.setKey(key);
filter.setUserAttribute(attrs);
```

should be replaced with:

```java
AnonymousAuthenticationFilter filter =
                new AnonymousAuthenticationFilter(key,attrs.getPassword(),attrs.getAuthorities());
```

and

```xml
<b:bean class="org.springframework.security.web.authentication.AnonymousAuthenticationFilter">
        <b:property name="key" value="key"/>
        <b:property name="userAttribute" ref="userAttribute"/>
</b:bean>
```

can be replaced with

```xml
<b:bean class="org.springframework.security.web.authentication.AnonymousAuthenticationFilter">
        <b:constructor-arg value="key"/>
        <b:constructor-arg value="#{userAttribute.password}"/>
        <b:constructor-arg value="#{userAttribute.authorities}"/>
</b:bean>
```

4.7.5. LoginUrlAuthenticationEntryPoint

The `LoginUrlAuthenticationEntryPoint` default constructor and the `setLoginFormUrl` method was removed in favor of constructor injection. For example:

```java
LoginUrlAuthenticationEntryPoint entryPoint = new LoginUrlAuthenticationEntryPoint();
entryPoint.setLoginFormUrl("/login");
```

should be replaced with

```
LoginUrlAuthenticationEntryPoint entryPoint = new LoginUrlAuthenticationEntryPoint(loginFormUrl);
```
JAVA

and

```
<b:bean class="org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint">
        <b:property name="loginFormUrl" value="/login"/>
</b:bean>
```
XML

should be replaced with:

```
<b:bean class="org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint">
        <b:constructor-arg value="/login"/>
</b:bean>
```
XML

4.7.6. PreAuthenticatedGrantedAuthoritiesUserDetailsService

`PreAuthenticatedGrantedAuthoritiesUserDetailsService` removed `createuserDetails` in favor of `createUserDetails`.

The new method has a correction in the case (i.e. U instead of u).

This means if you have a subclass of `PreAuthenticatedGrantedAuthoritiesUserDetailsService` that overrides `createuserDetails`

```
public class SubclassPreAuthenticatedGrantedAuthoritiesUserDetailsService extends
PreAuthenticatedGrantedAuthoritiesUserDetailsService {

        @Override
        protected UserDetails createuserDetails(Authentication token,
                        Collection<? extends GrantedAuthority> authorities) {
                // customize
        }
}
```
JAVA

it should be changed to override `createUserDetails`

```
public class SubclassPreAuthenticatedGrantedAuthoritiesUserDetailsService extends
PreAuthenticatedGrantedAuthoritiesUserDetailsService {

        @Override
        protected UserDetails createUserDetails(Authentication token,
                        Collection<? extends GrantedAuthority> authorities) {
                // customize
        }
}
```
JAVA

4.7.7. AbstractRememberMeServices

`AbstractRememberMeServices` and its subclasses `PersistentTokenBasedRememberMeServices` and `TokenBasedRememberMeServices` removed the default constructor and the `setKey` and `setUserDetailsService` methods in favor of constructor injection.

4.7.8. PersistentTokenBasedRememberMeServices

`AbstractRememberMeServices` and its subclasses `PersistentTokenBasedRememberMeServices` and `TokenBasedRememberMeServices` removed the default constructor and the `setKey` and `setUserDetailsService` methods in favor of constructor injection. For example:

```
PersistentTokenBasedRememberMeServices services = new PersistentTokenBasedRememberMeServices();
services.setKey(key);
services.setUserDetailsService(userDetailsService);
services.setTokenRepository(tokenRepository);
```
JAVA

should be replaced with

```java
PersistentTokenBasedRememberMeServices services =
                new PersistentTokenBasedRememberMeServices(key, userDetailsService, tokenRepository);
```

and

```xml
<b:bean class="org.springframework.security.web.authentication.rememberme.PersistentTokenBasedRememberMeServices">
        <b:property name="key" value="key"/>
        <b:property name="userDetailsService" ref="userDetailsService"/>
        <b:property name="tokenRepository" ref="tokenRepository"/>
</b:bean>
```

should be replaced with:

```xml
<b:bean class="org.springframework.security.web.authentication.rememberme.PersistentTokenBasedRememberMeServices">
        <b:constructor-arg value="key"/>
        <b:constructor-arg ref="userDetailsService"/>
        <b:constructor-arg ref="tokenRepository"/>
</b:bean>
```

4.7.9. RememberMeAuthenticationFilter

`RememberMeAuthenticationFilter` default constructor and the `setAuthenticationManager` and `setRememberMeServices` methods were removed in favor of constructor injection.

```java
RememberMeAuthenticationFilter filter = new RememberMeAuthenticationFilter();
filter.setAuthenticationManager(authenticationManager);
filter.setRememberMeServices(rememberMeServices);
```

should be replaced with

```java
RememberMeAuthenticationFilter filter =
                new RememberMeAuthenticationFilter(authenticationManager,rememberMeServices);
```

and

```xml
<b:bean class="org.springframework.security.web.authentication.rememberme.RememberMeAuthenticationFilter">
        <b:property name="authenticationManager" ref="authenticationManager"/>
        <b:property name="rememberMeServices" ref="rememberMeServices"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.authentication.rememberme.RememberMeAuthenticationFilter">
        <b:constructor-arg ref="authenticationManager"/>
        <b:constructor-arg ref="rememberMeServices"/>
</b:bean>
```

4.7.10. TokenBasedRememberMeServices

`AbstractRememberMeServices` and its subclasses `PersistentTokenBasedRememberMeServices` and `TokenBasedRememberMeServices` removed the default constructor and the `setKey` and `setUserDetailsService` methods in favor of constructor injection. For example:

```java
TokenBasedRememberMeServices services = new TokenBasedRememberMeServices();
services.setKey(key);
services.setUserDetailsService(userDetailsService);
```

should be replaced with

```java
TokenBasedRememberMeServices services =
                new TokenBasedRememberMeServices(key, userDetailsService);
```

and

```xml
<b:bean class="org.springframework.security.web.authentication.rememberme.TokenBasedRememberMeServices">
        <b:property name="key" value="key"/>
        <b:property name="userDetailsService" ref="userDetailsService"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.authentication.rememberme.TokenBasedRememberMeServices">
        <b:constructor-arg value="key"/>
        <b:constructor-arg ref="userDetailsService"/>
</b:bean>
```

4.7.11. ConcurrentSessionControlStrategy

`ConcurrentSessionControlStrategy` was replaced with `ConcurrentSessionControlAuthenticationStrategy`. Previously `ConcurrentSessionControlStrategy` could not be decoupled from `SessionFixationProtectionStrategy`. Now it is completely decoupled. For example, the following:

```java
ConcurrentSessionControlStrategy strategy = new ConcurrentSessionControlStrategy(sessionRegistry);
```

can be replaced with

```java
List<SessionAuthenticationStrategy> delegates = new ArrayList<SessionAuthenticationStrategy>();
delegates.add(new ConcurrentSessionControlAuthenticationStrategy(sessionRegistry));
delegates.add(new SessionFixationProtectionStrategy());
delegates.add(new RegisterSessionAuthenticationStrategy(sessionRegistry));
CompositeSessionAuthenticationStrategy strategy = new CompositeSessionAuthenticationStrategy(delegates);
```

and

```xml
<b:bean class="org.springframework.security.web.authentication.session.ConcurrentSessionControlStrategy">
        <b:constructor-arg ref="sessionRegistry"/>
</b:bean>
```

can be replaced with

```xml
<b:bean class="org.springframework.security.web.authentication.session.CompositeSessionAuthenticationStrategy">
        <b:constructor-arg>
                <b:list>
                        <b:bean
class="org.springframework.security.web.authentication.session.ConcurrentSessionControlAuthenticationStrategy">
                                <b:constructor-arg ref="sessionRegistry"/>
                        </b:bean>
                        <b:bean
class="org.springframework.security.web.authentication.session.SessionFixationProtectionStrategy"/>
                        <b:bean
class="org.springframework.security.web.authentication.session.RegisterSessionAuthenticationStrategy">
                                <b:constructor-arg ref="sessionRegistry"/>
                        </b:bean>
                </b:list>
        </b:constructor-arg>
</b:bean>
```

4.7.12. SessionFixationProtectionStrategy

`SessionFixationProtectionStrategy` removed `setRetainedAttributes` method in favor of users subclassing `SessionFixationProtectionStrategy` and overriding `extractAttributes` method. This means the following:

```java
SessionFixationProtectionStrategy strategy = new SessionFixationProtectionStrategy();
strategy.setRetainedAttributes(attrsToRetain);
```

should be replaced with

```java
public class AttrsSessionFixationProtectionStrategy extends SessionFixationProtectionStrategy {
	private final Collection<String> attrsToRetain;

	public AttrsSessionFixationProtectionStrategy(
			Collection<String> attrsToRetain) {
		this.attrsToRetain = attrsToRetain;
	}

	@Override
	protected Map<String, Object> extractAttributes(HttpSession session) {
		Map<String,Object> attrs = new HashMap<String, Object>();
		for(String attr : attrsToRetain) {
			attrs.put(attr, session.getAttribute(attr));
		}
		return attrs;
	}

}

SessionFixationProtectionStrategy strategy = new AttrsSessionFixationProtectionStrategy(attrsToRetain);
```

4.7.13. BasicAuthenticationFilter

`BasicAuthenticationFilter` default constructor and the `setAuthenticationManager` and `setRememberMeServices` methods were removed in favor of constructor injection.

```java
BasicAuthenticationFilter filter = new BasicAuthenticationFilter();
filter.setAuthenticationManager(authenticationManager);
filter.setAuthenticationEntryPoint(entryPoint);
filter.setIgnoreFailure(true);
```

should be replaced with

```java
BasicAuthenticationFilter filter =
		new BasicAuthenticationFilter(authenticationManager,entryPoint);
```

Using this constructor automatically sets ignoreFalure to true

and

```xml
<b:bean class="org.springframework.security.web.authentication.www.BasicAuthenticationFilter">
	<b:property name="authenticationManager" ref="authenticationManager"/>
	<b:property name="authenticationEntryPoint" ref="entryPoint"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.authentication.www.BasicAuthenticationFilter">
	<b:constructor-arg ref="authenticationManager"/>
	<b:constructor-arg ref="entryPoint"/>
</b:bean>
```

4.7.14. SecurityContextPersistenceFilter

`SecurityContextPersistenceFilter` removed the `setSecurityContextRepository` in favor of constructor injection. For example:

```java
SecurityContextPersistenceFilter filter = new SecurityContextPersistenceFilter();
filter.setSecurityContextRepository(securityContextRepository);
```

should be replaced with

```java
SecurityContextPersistenceFilter filter = new SecurityContextPersistenceFilter(securityContextRepository);
```

and

```xml
<b:bean class="org.springframework.security.web.context.SecurityContextPersistenceFilter">
        <b:property name="securityContextRepository" ref="securityContextRepository"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.context.SecurityContextPersistenceFilter">
        <b:constructor-arg ref="securityContextRepository"/>
</b:bean>
```

4.7.15. RequestCacheAwareFilter

`RequestCacheAwareFilter` removed the `setRequestCache` in favor of constructor injection. For example:

```java
RequestCacheAwareFilter filter = new RequestCacheAwareFilter();
filter.setRequestCache(requestCache);
```

should be replaced with

```java
RequestCacheAwareFilter filter = new RequestCacheAwareFilter(requestCache);
```

and

```xml
<b:bean class="org.springframework.security.web.savedrequest.RequestCacheAwareFilter">
        <b:property name="requestCache" ref="requestCache"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.savedrequest.RequestCacheAwareFilter">
        <b:constructor-arg ref="requestCache"/>
</b:bean>
```

4.7.16. ConcurrentSessionFilter

`ConcurrentSessionFilter` removed the default constructor and the `setExpiredUrl` and `setSessionRegistry` methods in favor of constructor injection. For example:

```java
ConcurrentSessionFilter filter = new ConcurrentSessionFilter();
filter.setSessionRegistry(sessionRegistry);
filter.setExpiredUrl("/expired");
```

should be replaced with

```java
ConcurrentSessionFilter filter = new ConcurrentSessionFilter(sessionRegistry,"/expired");
```

and

```xml
<b:bean class="org.springframework.security.web.session.ConcurrentSessionFilter">
        <b:property name="sessionRegistry" ref="sessionRegistry"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.session.ConcurrentSessionFilter">
        <b:constructor-arg ref="sessionRegistry"/>
</b:bean>
```

4.7.17. SessionManagementFilter

`SessionManagementFilter` removed the `setSessionAuthenticationStrategy` method in favor of constructor injection. For example:

```java
SessionManagementFilter filter = new SessionManagementFilter(securityContextRepository);
filter.setSessionAuthenticationStrategy(sessionAuthenticationStrategy);
```

should be replaced with

```java
SessionManagementFilter filter = new SessionManagementFilter(securityContextRepository, sessionAuthenticationStrategy);
```

and

```xml
<b:bean class="org.springframework.security.web.session.SessionManagementFilter">
        <b:constructor-arg ref="securityContextRepository"/>
        <b:property name="sessionAuthenticationStrategy" ref="sessionAuthenticationStrategy"/>
</b:bean>
```

should be replaced with

```xml
<b:bean class="org.springframework.security.web.session.SessionManagementFilter">
        <b:constructor-arg ref="securityContextRepository"/>
        <b:constructor-arg ref="sessionAuthenticationStrategy"/>
</b:bean>
```

4.7.18. RequestMatcher

The `RequestMatcher` and its implementations have moved from the package `org.springframework.security.web.util` to `org.springframework.security.web.util.matcher`. Specifically

- `org.springframework.security.web.util.RequestMatcher` → `org.springframework.security.web.util.matcher.RequestMatcher`
- `org.springframework.security.web.util.AntPathRequestMatcher` → `org.springframework.security.web.util.matcher.AntPathRequestMatcher`
- `org.springframework.security.web.util.AnyRequestMatcher` → `org.springframework.security.web.util.matcher.AnyRequestMatcher.INSTANCE`
- `org.springframework.security.web.util.ELRequestMatcher` → `org.springframework.security.web.util.matcher.ELRequestMatcher`
- `org.springframework.security.web.util.IpAddressMatcher` → `org.springframework.security.web.util.matcher.IpAddressMatcher`
- `org.springframework.security.web.util.RequestMatcherEditor` → `org.springframework.security.web.util.matcher.RequestMatcherEditor`
- `org.springframework.security.web.util.RegexRequestMatcher` → `org.springframework.security.web.util.matcher.RegexRequestMatcher`

4.7.19. WebSecurityExpressionHandler

`WebSecurityExpressionHandler` was removed in favor of using `SecurityExpressionHandler<FilterInvocation>`.

This means if you are using:

```java
WebSecurityExpressionHandler handler = ...
```

it needs to be updated to

```java
SecurityExpressionHandler<FilterInvocation> handler = ...
```

If you implement WebSecurityExpressionHandler:

```java
public class CustomWebSecurityExpressionHandler implements WebSecurityExpressionHandler {
        ...
}
```

then it must be updated to:

```java
public class CustomWebSecurityExpressionHandler implements SecurityExpressionHandler<FilterInvocation> {
        ...
}
```

5. Update Spring Security

Now you can update to Spring Security 4.x. If you are using Maven and Spring Security's BOM, you can do something like this:

```xml
<dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-bom</artifactId>
			<version>4.0.0.RELEASE</version>
			<type>pom</type>
			<scope>import</scope>
		</dependency>
	</dependencies>
</dependencyManagement>
```

Now all of the Spring Security dependencies that do not specify a version will use the updated Spring Security version.

Alternatively, you can update each of the Spring Security dependencies within your pom. For example, the following would update spring-security-core to use version 4.0.0.RELEASE

```xml
<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-core</artifactId>
	<version>4.0.0.RELEASE</version>
</dependency>
```

6. Migrate XML Namespace Defaults

We updated the default values for many of the Spring Security XML Namespace Elements. You can find a detailed list of changes and how to address them below.

If you do not use XML based configuration, you may safely skip this section and proceed to Migrate Default Filter URLs

6.1. Related Links

For thoroughness we have include the related links in the table below.

JIRA	Commits
SEC-2783 (https://jira.spring.io/browse/SEC-2783)	c67ff42 (https://github.com/spring-projects/spring-security/commit/c67ff42b8abe124b7956896c78e9aac896fd79d9)
SEC-2347 (https://jira.spring.io/browse/SEC-2347)	4392205 (https://github.com/spring-projects/spring-security/commit/4392205f63e49b9675b06e584f571a48b017d0b6)
SEC-2348 (https://jira.spring.io/browse/SEC-2348)	eedbf44 (https://github.com/spring-projects/spring-security/commit/eedbf442359f9a99e367f2fdef61deea1cef46c9)
SEC-2873 (https://jira.spring.io/browse/SEC-2873)	5f57e5b (https://github.com/spring-projects/spring-security/commit/5f57e5b0c3726466db4f5d0521ac26423f0d9cd4)
SEC-2916 (https://jira.spring.io/browse/SEC-2916)	c94a5cf (https://github.com/spring-projects/spring-security/commit/c94a5cf8e268a4e8090c28268372d7d61c367028)

6.2. Migrate <http>

The http@use-expressions (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-http-use-expressions) attribute's default value changed from false to true. This means if the use-expression attribute is not explicitly configured, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

Spring Security 3.2.x Sample Configuration

```xml
<http> (1)
        <intercept-url pattern="/login" access="ROLE_ANONYMOUS"/>
        <intercept-url pattern="/**" access="ROLE_USER"/>
        ...
</http>
```

(1) Observe that the use-expressions attribute is not provided. If it were provided, then nothing needs to be done.

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

Migration to Spring Security 4 Configuration

```xml
<http use-expressions="false"> (1)
        <intercept-url pattern="/login" access="ROLE_ANONYMOUS"/>
        <intercept-url pattern="/**" access="ROLE_USER"/>
        ...
</http>
```

(1) We explicitly provide the use-expressions attribute. Again, if the attribute was already provided, then nothing needs to be done.

Alternatively, the application can omit the the use-expressions attribute and switch to using expressions. For example, something similar to the following:

Alternative Migration to Spring Security 4 Configuration

```xml
<http>
        <intercept-url pattern="/login" access="permitAll"/>
        <intercept-url pattern="/**" access="hasRole('USER')"/>
        ...
</http>
```

The http@disable-url-rewriting (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-http-disable-url-rewriting) attribute's default value changed from false to true.

It is recommended to disable url rewriting to prevent the JSESSIONID from being included in URLs. If your application does not use url rewriting, then it is preferrable to leave this attribute set to the default value.

This means if the disable-url-rewriting attribute is not explicitly configured and you are relying on url rewriting, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

Spring Security 3.2.x Sample Configuration

```xml
<http> (1)
        ...
</http>
```

(1) Observe that the disable-url-rewriting attribute is not provided. If it were provided, then nothing needs to be done.

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

Migration to Spring Security 4 Configuration

```xml
<http disable-url-rewriting="false"> (1)
        ...
</http>
```

(1) We explicitly provide the disable-url-rewriting attribute. Again, if the attribute was already provided, then nothing needs to be done.

6.3. Migrating <form-login>

If the `<form-login>` is being used within an application, then some of the default attributes have changed. Below are detailed description of the changes and how to migrate:

- The form-login@username-parameter (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-form-login-username-parameter) attribute default value changed from j_username to username. If an application explicitly provides the attribute, no action is required for the migration.
- The form-login@password-parameter (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-form-login-password-parameter) attribute default value changed from j_password to password. If an application explicitly provides the attribute, no action is required for the migration.
- The form-login@login-processing-url (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-form-login-login-processing-url) attribute default value changed from /j_spring_security_check to POST /login. If an application explicitly provides the attribute, no action is required for the migration.
- The form-login@authentication-failure-url (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-form-login-authentication-failure-url) attribute default value changed from appending ?login_error to the login-page to appending ?error to the login-page. If an application explicitly provides the attribute, no action is required for the migration.

These changes mean if you have the following configuration within your XML configuration when using Spring Security 3.2.x:

Spring Security 3.2.x Sample Configuration

```xml
<http>
        ...
        <form-login login-page="/login"/>
</http>
```

You will need to migrate by explicitly configuring the attributes that have new default values when migrating to Spring Security 4.x:

Any attribute that is already explicitly provided will not be impacted and requires no action.

Migration to Spring Security 4 Configuration

```
<http>
                ...
                <form-login login-page="/login"
                                username-parameter="j_username" (1)
                                password-parameter="j_password" (2)
                                login-processing-url="/j_spring_security_check" (3)
                                authentication-failure-url="/login?login_error=1" (4)
                />
</http>
```

(1) If the configuration does not specify the username-parameter, then it should be explicitly stated

(2) If the configuration does not specify the password-parameter, then it should be explicitly stated

(3) If the configuration does not specify the login-processing-url, then it should be explicitly stated

(4) If the configuration does not specify the authentication-failure-url, then it should be explicitly stated

Alternatively, the application can be updated to use the new defaults. For example, one might update their log in form to look like the following:

Alternative Migration to Spring Security 4.x (i.e. login.jsp)

```
<c:if test="${param.error != null}"> (1)
        <p>Invalid username / password</p>
</c:if>
<c:url var="loginUrl" value="/login"/> (2)
<form action="${loginUrl}" method="post">
        <p><label for="username">User:</label></p>
        <input type="text" id="username" name="username"/> (3)

        <p><label for="password">Password:</label></p>
        <input type="password" id="password" name="password"> (4)

        <div>
                <input name="submit" type="submit"/>
        </div>
</form>
```

(1) If the configuration does not specify the authentication-failure-url, then detect that an invalid log in check to see if the HTTP parameter error is not null.

(2) If the configuration does not specify the login-processing-url, then modify the URL to submit to be "/login"

(3) If the configuration does not specify the username-parameter, then modify the username HTTP parameter to be "username"

(4) If the configuration does not specify the password-parameter, then modify the password HTTP parameter to be "password"

6.4. Migrating <logout>

If the `<logout>` is being used within an application, then some of the default attributes have changed. Below are detailed description of the changes and how to migrate:

- The logout@logout-url (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-logout-logout-url) attribute default value changed from "/j_spring_security_logout" to "/logout". If an application explicitly provides the attribute, no action is required for the migration.

These changes mean if you have the following configuration within your XML configuration when using Spring Security 3.2.x:

Spring Security 3.2.x Sample Configuration

```xml
<http>
        ...
        <logout/>
</http>
```

You will need to migrate by explicitly configuring the logout-url attribute when migrating to Spring Security 4.x:

If the logout-url attribute is already explicitly provided the application will not be impacted and no action is required.

Migration to Spring Security 4 Configuration

```xml
<http>
                ...
                <logout logout-url="/j_spring_security_logout"/> (1)
                />
</http>
```

(1) If the configuration does not specify the logout-url attribute, then it should be explicitly stated

Alternatively, the application can be updated to use the new defaults.

6.5. Migrating <openid-login>

The openid-login@login-processing-url (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-openid-login-login-processing-url) attribute default value changed from /j_spring_openid_security_check to /login/openid.

This means if the login-processing-url attribute is not explicitly configured, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

Spring Security 3.2.x Sample Configuration

```xml
<http>
        <openid-login /> (1)
        ...
</http>
```

(1) Observe that the login-processing-url attribute is not provided. If it were provided, then nothing needs to be done.

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

Migration to Spring Security 4 Configuration

```xml
<http>
        <openid-login login-processing-url="/j_spring_openid_security_check"/> (1)
        ...
</http>
```

(1) We explicitly provide the login-processing-url attribute. Again, if the attribute was already provided, then nothing needs to be done.

Alternatively, the application can omit the the login-processing-url attribute and update the log in form. For example, something similar to the following:

Alternative Migration to Spring Security 4.x (i.e. login.jsp)

```xml
<c:url var="openidLoginUrl" value="/login/openid"/> (1)
<form action="${openidLoginUrl}" method="post">
        <div>
                <input name="openid_identifier" type="text" value="http://" />
                <input type="submit" value="Sign-In"/>
        </div>
</form>
```

(1) If the configuration does not specify the login-processing-url attribute, then update the log in action to "/login/openid".

6.6. Migrating <headers>

As Spring Security 4.0+ Security HTTP Response Headers is now enabled by default. This means if an application did not provide the headers (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-headers) element, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

Spring Security 3.2.x Sample Configuration

```xml
<http>
        ...
        <!-- no headers element -->
</http>
```

The application will need updated. The quickest, but not ideal, solution is to explicitly disable the headers protection using headers@disabled (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-headers-disabled). For example:

Migration to Spring Security 4 Configuration

```xml
<http>
        ...
        <headers disabled="true"/>
</http>
```

Alternatively, the application would enable Security HTTP Response Headers. In many instances, leaving the Security HTTP Response Headers enabled will not have a negative impact on an application.

Strict Transport Security (http://docs.spring.io/spring-security/site/docs/current/reference/html/headers.html#headers-hsts) will cause infinite redirects if anywhere within your domain forcefully redirects from HTTPS to HTTP for a subset of pages. If your domain forcfully redirects to HTTP when HTTPS is requested, you will need to ensure to remove the redirect (recommended) or disable Strict Transport Security.

Developers are encouraged to read Security HTTP Response Headers for details on using this feature.

6.7. Migrating <csrf>

As Spring Security 4.0+ CSRF Protection is now enabled by default. This means if an application did not provide the csrf (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-csrf) element, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

```xml
<http>
        ...
        <!-- no csrf element -->
</http>
```

The application will need updated. The quickest, but not ideal, solution is to explicitly disable the csrf protection using csrf@disabled (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-csrf-disabled). For example:

Migration to Spring Security 4 Configuration

```xml
<http>
        ...
        <csrf disabled="true"/>
</http>
```

Alternatively, the application would enable CSRF. For more details refer to Using Spring Security CSRF Protection (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#csrf-using)

6.8. Migrating <remember-me>

If the `<remember-me>` element is being used within an application, then some of the default attributes have changed. Below are detailed description of the changes and how to migrate:

- The remember-me@remember-me-parameter (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-remember-me-remember-me-parameter) attribute default value changed from "_spring_security_remember_me" to "remember-me". If an application explicitly provides the attribute, no action is required for the migration.
- The remember-me@remember-me-cookie (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-remember-me-remember-me-cookie) attribute default value changed from "SPRING_SECURITY_REMEMBER_ME_COOKIE" to "remember-me". If an application explicitly provides the

attribute, no action is required for the migration.

These changes mean if you have the following configuration within your XML configuration when using Spring Security 3.2.x:

```
<http>
        ...
        <remember-me />
</http>
```

You will need to migrate by explicitly configuring the attributes that have new default values when migrating to Spring Security 4.x:

Any attribute that is already explicitly provided will not be impacted and requires no action.

```
<http>
        ...
        <remember-me
                remember-me-parameter="_spring_security_remember_me" (1)
                remember-me-cookie="SPRING_SECURITY_REMEMBER_ME_COOKIE" (2)
        />
</http>
```

(1) If the configuration does not specify the remember-me-parameter, then it should be explicitly stated

(2) If the configuration does not specify the remember-me-cookie, then it should be explicitly stated

Alternatively, the application can be updated to use the new defaults. For example, one might update their log in form to look like the following:

login.html

```
<c:url var="loginUrl" value="/login"/>
<form action="${loginUrl}" method="post">
        ...

        <p><label for="remember-me">Remember Me</label></p>
        <input type="checkbox" id="remember-me" name="remember-me"/> (1)

        <div>
                <input name="submit" type="submit"/>
        </div>
</form>
```

(1) If the configuration does not specify the remember-me-parameter, then update the HTTP parameter name to be remember-me

This approach means that previously remembered users will be forgotten since the remember me cookie name will change. If you are fine with users needing to authenticate again, then nothing is required. If you do not want users to authenticate, then the cookie name must be set to SPRING_SECURITY_REMEMBER_ME_COOKIE as illustrated above.

6.9. Migrating <filter-security-metadata-source>

The filter-security-metadata-source@use-expressions (http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#nsa-filter-security-metadata-source-use-expressions) attribute's default value changed from false to true. This means if the use-expression attribute is not explicitly configured, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

Spring Security 3.2.x Sample Configuration

```
<filter-security-metadata-source> (1)
        <intercept-url pattern="/login" access="ROLE_ANONYMOUS"/>
        <intercept-url pattern="/**" access="ROLE_USER"/>
        ...
</filter-security-metadata-source>
```

(1) Observe that the use-expressions attribute is not provided. If it were provided, then nothing needs to be done.

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

Migration to Spring Security 4 Configuration

```xml
<filter-security-metadata-source use-expressions="false"> (1)
        <intercept-url pattern="/login" access="ROLE_ANONYMOUS"/>
        <intercept-url pattern="/**" access="ROLE_USER"/>
        ...
</filter-security-metadata-source>
```

(1) We explicitly provide the use-expressions attribute. Again, if the attribute was already provided, then nothing needs to be done.

Alternatively, the application can omit the the use-expressions attribute and switch to using expressions. For example, something similar to the following:

Alternative Migration to Spring Security 4 Configuration

```xml
<filter-security-metadata-source>
        <intercept-url pattern="/login" access="permitAll"/>
        <intercept-url pattern="/**" access="hasRole('USER')"/>
        ...
</filter-security-metadata-source>
```

7. Migrate Default Filter URLs

A number of servlet Filter's had their default URLs switched to help guard against information leakage.

7.1. Related Links

For thoroughness we have include the related links in the table below.

JIRA	Commits
SEC-2783 (https://jira.spring.io/browse/SEC-2783)	c67ff42 (https://github.com/spring-projects/spring-security/commit/c67ff42b8abe124b7956896c78e9aac896fd79d9)

7.2. CasAuthenticationFilter

The `CasAuthenticationFilter` filterProcessesUrl property default value changed from "/j_spring_cas_security_check" to "/login/cas". This means if the filterProcessesUrl property is not explicitly specified, then the configuration will need updated. For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

```
XML
<b:bean id="casFilter"
                class="org.springframework.security.cas.web.CasAuthenticationFilter">
        <b:property name="authenticationManager" ref="authenticationManager"/>
</b:bean>
```

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

```
XML
<b:bean id="casFilter"
                class="org.springframework.security.cas.web.CasAuthenticationFilter">
        <b:property name="authenticationManager" ref="authenticationManager"/>
        <b:property name="filterProcessesUrl" value="/j_spring_cas_security_check"/>
</b:bean>
```

Alternatively, the `ServiceProperties` can be updated to use the new default:

```
XML
<bean id="serviceProperties"
                class="org.springframework.security.cas.ServiceProperties">
        <property name="service"
                        value="https://example.com/cas-sample/login/cas"/>
</bean>
```

7.3. SwitchUserFilter

- The `SwitchUserFilter` switchUserUrl property default value changed from "/j_spring_security_switch_user" to "/login/impersonate". This means if the switchUserUrl property is not explicitly specified, then the configuration will need updated.
- The `SwitchUserFilter` exitUserUrl property default value changed from "/j_spring_security_exit_user" to "/logout/impersonate". This means if the exitUserUrl property is not explicitly specified, then the configuration will need updated.

For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

```
XML
<b:bean id="switchUserProcessingFilter"
class="org.springframework.security.web.authentication.switchuser.SwitchUserFilter">
        <b:property name="userDetailsService" ref="userDetailsService" />
        <b:property name="targetUrl" value="/" />
</b:bean>
```

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

```xml
<b:bean id="switchUserProcessingFilter"
class="org.springframework.security.web.authentication.switchuser.SwitchUserFilter">
        <b:property name="switchUserUrl" value="/j_spring_security_switch_user" />
        <b:property name="exitUserUrl" value="/j_spring_security_exit_user" />

        <b:property name="userDetailsService" ref="userDetailsService" />
        <b:property name="targetUrl" value="/" />
</bean>
```

Alternatively, the URL's within the application can be updated from:

- "/j_spring_security_switch_user" to "/login/impersonate"
- "/j_spring_security_exit_user" to "/logout/impersonate"

7.4. LogoutFilter

The `LogoutFilter` filterProcessesUrl property default value changed from "/j_spring_security_logout" to "/logout". This means if the filterProcessesUrl property is not explicitly specified, then the configuration will need updated.

For example, if an application using Spring Security 3.2.x contains a configuration similar to the following:

```xml
<b:bean id="logoutFilter" class="org.springframework.security.web.authentication.logout.LogoutFilter">
        <b:constructor-arg value="/logout-success"/>
        <b:constructor-arg>
                <b:bean class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler"/>
        </b:constructor-arg>
</b:bean>
```

The configuration will need to be updated to something similar to the following when Spring Security 4.x:

```xml
<b:bean id="logoutFilter" class="org.springframework.security.web.authentication.logout.LogoutFilter">
        <b:constructor-arg value="/logout-success"/>
        <b:constructor-arg>
                <b:bean class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler"/>
        </b:constructor-arg>


        <b:property name="filterProcessesUrl" value="/j_spring_security_logout"/>
</b:bean>
```

Alternatively, the URL's within the application can be updated from "/j_spring_security_logout" to "/logout".

8. Header Configuration Changes

In Spring Security 3.x the HTTP Response Header configuration was difficult to customize. If an application overrode a single default, then all of the other defaults would be disabled. This was unintuitive, error prone, and most importantly not very secure.

Spring Security 4.x has changed both the Java Configuration and XML Configuration to require explicit disabling of defaults. Additionally, it has made customizing a single default much easier.

If an application has customized the HTTP Response Header Configuration in any way, they are impacted by this change. If the application used the defaults, then they are not impacted by this change.

A detailed description of how to configure Security HTTP Response Headers can be found in the reference. Below we highlight the changes in configuring the Security HTTP Response Headers between 3.x and 4.x.

- Migrating XML Based Configuration
- Migrating Java Based Configuration

8.1. Related Links

For thoroughness we have include the related links in the table below.

JIRA	Commits
SEC-2348 (https://jira.spring.io/browse/SEC-2348)	eedbf44 (https://github.com/spring-projects/spring-security/commit/eedbf442359f9a99e367f2fdef61deea1cef46c9)

8.2. Header Samples

In Spring Security 3.x, the following configuration

```xml
<http>
	<headers>
		<frame-options policy="SAMEORIGIN"/>
	</headers>

	...
</http>
```

would add the following header:

```http
X-Frame-Options: SAMEORIGIN
```

In Spring Security 4.x, the same configuration would add

```http
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
```

If we want to the configuration the same, we must explicitly disable the other defaults.

```xml
<http>
	...
	<headers defaults-disabled="true">
		<frame-options policy="SAMEORIGIN"/>
	</headers>
```

would add the following header:

```http
X-Frame-Options: SAMEORIGIN
```

9. Automatic ROLE_ prefixing

Spring Security 4 automatically prefixes any role with ROLE_. The changes were made as part of SEC-2758 (https://jira.spring.io/browse/SEC-2758)

9.1. Related Links

For thoroughness we have include the related links in the table below.

JIRA	Commits
SEC-2758 (https://jira.spring.io/browse/SEC-2758)	6627f76 (https://github.com/spring-projects/spring-security/commit/6627f76df7d93dfd85dd57954f11f595b1ab5f07)
SEC-2926 (https://jira.spring.io/browse/SEC-2926)	09acc2b (https://github.com/spring-projects/spring-security/commit/09acc2b7a531a5f3ded7cec1226a888441f78584)

9.2. ROLE_ Prefixing Passivity

Passivity is impacted if the application's users' roles are **not** prefixed with ROLE_. If all of the application's users' roles are prefixed with ROLE_ then it is NOT impacted.

9.3. Disable ROLE_ Prefixing

One can disable automatic ROLE_ prefixing using a `BeanPostProcessor` similar to the following:

JAVA

```
package sample.role_;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.core.PriorityOrdered;
import org.springframework.security.access.annotation.Jsr250MethodSecurityMetadataSource;
import org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler;
import org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler;
import org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter;

public class DefaultRolesPrefixPostProcessor implements BeanPostProcessor, PriorityOrdered {

	@Override
	public Object postProcessAfterInitialization(Object bean, String beanName)
			throws BeansException {

		// remove this if you are not using JSR-250
		if(bean instanceof Jsr250MethodSecurityMetadataSource) {
			((Jsr250MethodSecurityMetadataSource) bean).setDefaultRolePrefix(null);
		}

		if(bean instanceof DefaultMethodSecurityExpressionHandler) {
			((DefaultMethodSecurityExpressionHandler) bean).setDefaultRolePrefix(null);
		}
		if(bean instanceof DefaultWebSecurityExpressionHandler) {
			((DefaultWebSecurityExpressionHandler) bean).setDefaultRolePrefix(null);
		}
		if(bean instanceof SecurityContextHolderAwareRequestFilter) {
			((SecurityContextHolderAwareRequestFilter)bean).setRolePrefix("");
		}
		return bean;
	}

	@Override
	public Object postProcessBeforeInitialization(Object bean, String beanName)
			throws BeansException {
		return bean;
	}

	@Override
	public int getOrder() {
		return PriorityOrdered.HIGHEST_PRECEDENCE;
	}
}
```

and then defining it as a Bean:

```xml
<b:bean class="sample.role_.DefaultRolesPrefixPostProcessor"/>
```

Last updated 2015-11-28 18:53:45 PST