Spring Boot Auto Configuration

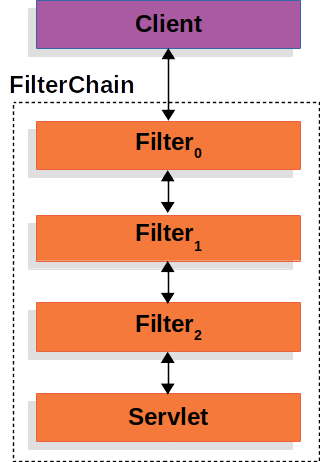
Spring Boot automatically:

* Enables Spring Security’s default configuration, which creates a servlet Filter as a bean named springSecurityFilterChain. This bean is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, and so on) within your application.
* Creates a UserDetailsService bean with a username of user and a randomly generated password that is logged to the console.
* Registers the Filter with a bean named springSecurityFilterChain with the Servlet container for every request.

Spring Boot is not configuring much, but it does a lot. A summary of the features follows:

* Require an authenticated user for any interaction with the application
* Generate a default login form for you
* Let the user with a username of user and a password that is logged to the console to authenticate with form-based authentication (in the preceding example, the password is 8e557245-73e2-4286-969a-ff57fe326336)
* Protects the password storage with BCrypt
* Lets the user log out
* [CSRF attack](https://en.wikipedia.org/wiki/Cross-site_request_forgery) prevention
* [Session Fixation](https://en.wikipedia.org/wiki/Session_fixation) protection
* Security Header integration
  + [HTTP Strict Transport Security](https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security) for secure requests
  + [X-Content-Type-Options](https://msdn.microsoft.com/en-us/library/ie/gg622941(v=vs.85).aspx) integration
  + Cache Control (can be overridden later by your application to allow caching of your static resources)
  + [X-XSS-Protection](https://msdn.microsoft.com/en-us/library/dd565647(v=vs.85).aspx) integration
  + X-Frame-Options integration to help prevent [Clickjacking](https://en.wikipedia.org/wiki/Clickjacking)
* Integrate with the following Servlet API methods:
  + [HttpServletRequest#getRemoteUser()](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#getRemoteUser())
  + [HttpServletRequest.html#getUserPrincipal()](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#getUserPrincipal())
  + [HttpServletRequest.html#isUserInRole(java.lang.String)](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#isUserInRole(java.lang.String))
  + [HttpServletRequest.html#login(java.lang.String, java.lang.String)](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#login(java.lang.String,%20java.lang.String))
  + [HttpServletRequest.html#logout()](https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#logout())

Spring Security’s Servlet support is based on Servlet Filters, so it is helpful to look at the role of Filters generally first. The picture below shows the typical layering of the handlers for a single HTTP request.



**Figure 1. FilterChain**

The client sends a request to the application, and the container creates a FilterChain which contains the Filters and Servlet that should process the HttpServletRequest based on the path of the request URI. In a Spring MVC application the Servlet is an instance of [DispatcherServlet](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-servlet). At most one Servlet can handle a single HttpServletRequest and HttpServletResponse. However, more than one Filter can be used to:

* Prevent downstream Filters or the Servlet from being invoked. In this instance the Filter will typically write the HttpServletResponse.
* Modify the HttpServletRequest or HttpServletResponse used by the downstream Filters and Servlet

The power of the Filter comes from the FilterChain that is passed into it.

**Example 47.**FilterChain**Usage Example**

**public** **void** **doFilter**(ServletRequest request, ServletResponse response, FilterChain chain) {

*// do something before the rest of the application*

chain.doFilter(request, response); *// invoke the rest of the application*

*// do something after the rest of the application*

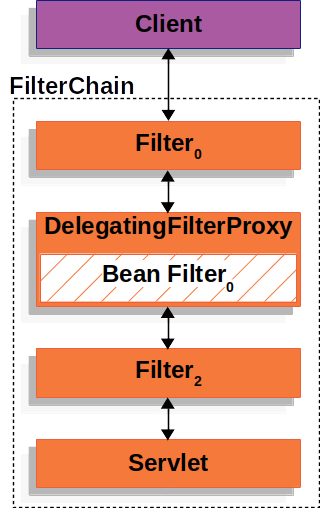
}

Since a Filter only impacts downstream Filters and the Servlet, the order each Filter is invoked is extremely important.

9.2. DelegatingFilterProxy

Spring provides a Filter implementation named [DelegatingFilterProxy](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/web/filter/DelegatingFilterProxy.html/) that allows bridging between the Servlet container’s lifecycle and Spring’s ApplicationContext. The Servlet container allows registering Filters using its own standards, but it is not aware of Spring defined Beans. DelegatingFilterProxy can be registered via standard Servlet container mechanisms, but delegate all the work to a Spring Bean that implements Filter.

Here is a picture of how DelegatingFilterProxy fits into the [Filters and the FilterChain](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-filters-review).



**Figure 2. DelegatingFilterProxy**

DelegatingFilterProxy looks up *Bean Filter0* from the ApplicationContext and then invokes *Bean Filter0*. The pseudo code of DelegatingFilterProxy can be seen below.

**Example 48.**DelegatingFilterProxy**Pseudo Code**

**public** **void** **doFilter**(ServletRequest request, ServletResponse response, FilterChain chain) {

*// Lazily get Filter that was registered as a Spring Bean*

*// For the example in* [*DelegatingFilterProxy*](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-delegatingfilterproxy-figure) *delegate is an instance of Bean Filter0*

Filter delegate = getFilterBean(someBeanName);

*// delegate work to the Spring Bean*

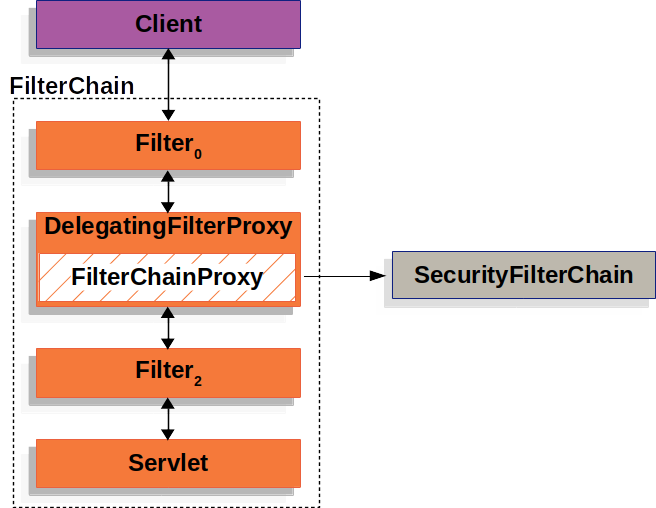
delegate.doFilter(request, response);

}

Another benefit of DelegatingFilterProxy is that it allows delaying looking Filter bean instances. This is important because the container needs to register the Filter instances before the container can startup. However, Spring typically uses a ContextLoaderListener to load the Spring Beans which will not be done until after the Filter instances need to be registered.

9.3. FilterChainProxy

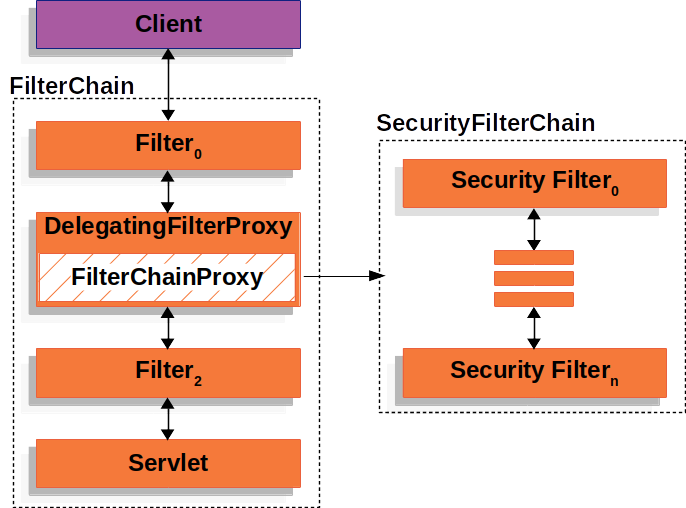
Spring Security’s Servlet support is contained within FilterChainProxy. FilterChainProxy is a special Filter provided by Spring Security that allows delegating to many Filter instances through [SecurityFilterChain](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-securityfilterchain). Since FilterChainProxy is a Bean, it is typically wrapped in a [DelegatingFilterProxy](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-delegatingfilterproxy).



**Figure 3. FilterChainProxy**

9.4. SecurityFilterChain

[SecurityFilterChain](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/SecurityFilterChain.html) is used by [FilterChainProxy](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-filterchainproxy) to determine which Spring Security Filters should be invoked for this request.



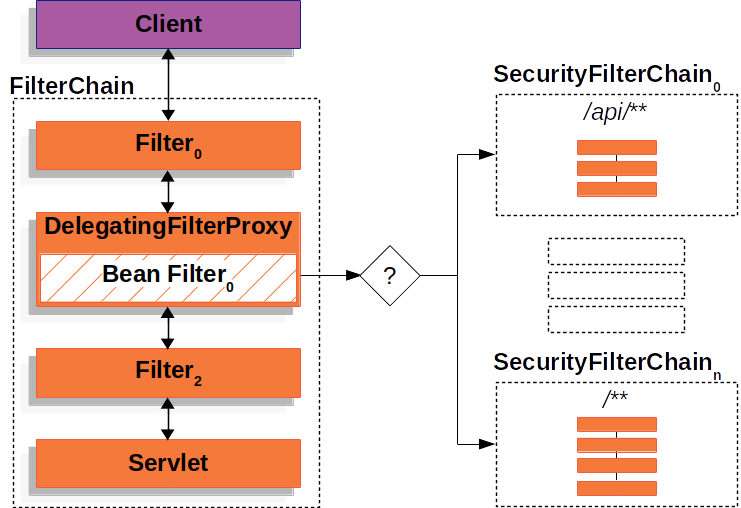
**Figure 4. SecurityFilterChain**

The [Security Filters](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-security-filters) in SecurityFilterChain are typically Beans, but they are registered with FilterChainProxy instead of [DelegatingFilterProxy](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-delegatingfilterproxy). FilterChainProxy provides a number of advantages to registering directly with the Servlet container or [DelegatingFilterProxy](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-delegatingfilterproxy). First, it provides a starting point for all of Spring Security’s Servlet support. For that reason, if you are attempting to troubleshoot Spring Security’s Servlet support, adding a debug point in FilterChainProxy is a great place to start.

Second, since FilterChainProxy is central to Spring Security usage it can perform tasks that are not viewed as optional. For example, it clears out the SecurityContext to avoid memory leaks. It also applies Spring Security’s [HttpFirewall](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-httpfirewall) to protect applications against certain types of attacks.

In addition, it provides more flexibility in determining when a SecurityFilterChain should be invoked. In a Servlet container, Filters are invoked based upon the URL alone. However, FilterChainProxy can determine invocation based upon anything in the HttpServletRequest by leveraging the RequestMatcher interface.

In fact, FilterChainProxy can be used to determine which SecurityFilterChain should be used. This allows providing a totally separate configuration for different *slices* if your application.



**Figure 5. Multiple SecurityFilterChain**

In the [Multiple SecurityFilterChain](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-multi-securityfilterchain-figure) Figure FilterChainProxy decides which SecurityFilterChain should be used. Only the first SecurityFilterChain that matches will be invoked. If a URL of /api/messages/ is requested, it will first match on SecurityFilterChain0's pattern of /api/\*\*, so only SecurityFilterChain0 will be invoked even though it also matches on SecurityFilterChainn. If a URL of /messages/ is requested, it will not match on SecurityFilterChain0's pattern of /api/\*\*, so FilterChainProxy will continue trying each SecurityFilterChain. Assuming that no other, SecurityFilterChain instances match SecurityFilterChainn will be invoked.

Notice that SecurityFilterChain0 has only three security Filters instances configured. However, SecurityFilterChainn has four security Filters configured. It is important to note that each SecurityFilterChain can be unique and configured in isolation. In fact, a SecurityFilterChain might have zero security Filters if the application wants Spring Security to ignore certain requests.

9.5. Security Filters

The Security Filters are inserted into the [FilterChainProxy](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-filterchainproxy) with the [SecurityFilterChain](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-securityfilterchain) API. The [order of Filter](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-filters-review)s matters. It is typically not necessary to know the ordering of Spring Security’s Filters. However, there are times that it is beneficial to know the ordering

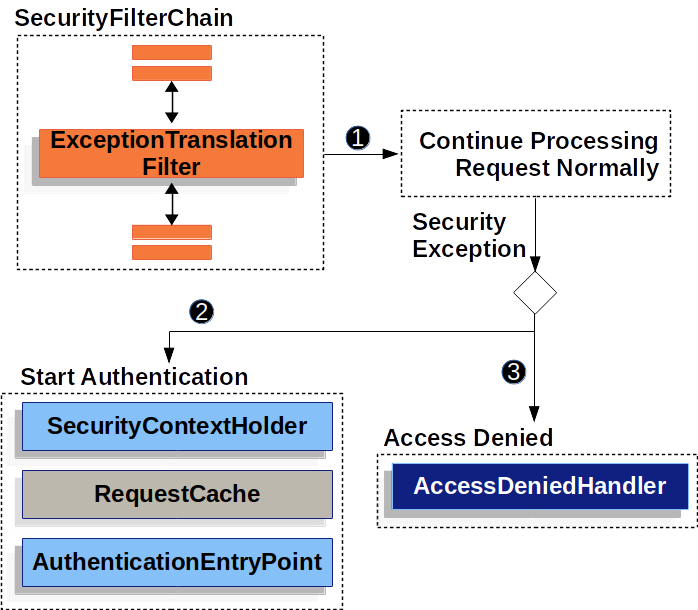
Below is a comprehensive list of Spring Security Filter ordering:

* ChannelProcessingFilter
* ConcurrentSessionFilter
* WebAsyncManagerIntegrationFilter
* SecurityContextPersistenceFilter
* HeaderWriterFilter
* CorsFilter
* CsrfFilter
* LogoutFilter
* OAuth2AuthorizationRequestRedirectFilter
* Saml2WebSsoAuthenticationRequestFilter
* X509AuthenticationFilter
* AbstractPreAuthenticatedProcessingFilter
* CasAuthenticationFilter
* OAuth2LoginAuthenticationFilter
* Saml2WebSsoAuthenticationFilter
* [UsernamePasswordAuthenticationFilter](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authentication-usernamepasswordauthenticationfilter)
* ConcurrentSessionFilter
* OpenIDAuthenticationFilter
* DefaultLoginPageGeneratingFilter
* DefaultLogoutPageGeneratingFilter
* [DigestAuthenticationFilter](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authentication-digest)
* BearerTokenAuthenticationFilter
* [BasicAuthenticationFilter](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authentication-basic)
* RequestCacheAwareFilter
* SecurityContextHolderAwareRequestFilter
* JaasApiIntegrationFilter
* RememberMeAuthenticationFilter
* AnonymousAuthenticationFilter
* OAuth2AuthorizationCodeGrantFilter
* SessionManagementFilter
* [ExceptionTranslationFilter](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-exceptiontranslationfilter)
* [FilterSecurityInterceptor](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authorization-filtersecurityinterceptor)
* SwitchUserFilter

9.6. Handling Security Exceptions

The [ExceptionTranslationFilter](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/access/ExceptionTranslationFilter.html) allows translation of [AccessDeniedException](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/access/AccessDeniedException.html) and [AuthenticationException](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/AuthenticationException.html) into HTTP responses.

ExceptionTranslationFilter is inserted into the [FilterChainProxy](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-filterchainproxy) as one of the [Security Filters](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-security-filters).



* number 1 First, the ExceptionTranslationFilter invokes FilterChain.doFilter(request, response) to invoke the rest of the application.
* number 2 If the user is not authenticated or it is an AuthenticationException, then *Start Authentication*.
  + The [SecurityContextHolder](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authentication-securitycontextholder) is cleared out
  + The HttpServletRequest is saved in the [RequestCache](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/savedrequest/RequestCache.html). When the user successfully authenticates, the RequestCache is used to replay the original request.
  + The AuthenticationEntryPoint is used to request credentials from the client. For example, it might redirect to a log in page or send a WWW-Authenticate header.
* number 3 Otherwise if it is an AccessDeniedException, then *Access Denied*. The AccessDeniedHandler is invoked to handle access denied.

|  |  |
| --- | --- |
|  | If the application does not throw an AccessDeniedException or an AuthenticationException, then ExceptionTranslationFilter does not do anything. |

The pseudocode for ExceptionTranslationFilter looks something like this:

**ExceptionTranslationFilter pseudocode**

**try** {

filterChain.doFilter(request, response);

} **catch** (AccessDeniedException | AuthenticationException e) {

**if** (!authenticated || e **instanceof** AuthenticationException) {

startAuthentication();

} **else** {

accessDenied();

}

}

|  |  |
| --- | --- |
|  | You will recall from [A Review of Filters](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-filters-review) that invoking FilterChain.doFilter(request, response) is the equivalent of invoking the rest of the application. This means that if another part of the application, (i.e. [FilterSecurityInterceptor](https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authorization-filtersecurityinterceptor) or method security) throws an AuthenticationException or AccessDeniedException it will be caught and handled here. |
|  | If the user is not authenticated or it is an AuthenticationException, then *Start Authentication*. |
|  | Otherwise, *Access Denied* |