Session 5 : Spring Security

sourcemind

What Is Spring Security ?

- Spring Security is a framework that provides **authentication**, **authorization**, and **protection against common attacks**. With first class support for securing both **imperative** and **reactive** applications, it is the de-facto standard for securing Spring-based applications.

https://docs.spring.io/spring-security/reference/index.html

Getting Spring Security

Spring Security versions are formatted as **MAJOR.MINOR.PATCH** such that:

- ✓ **MAJOR** versions may contain breaking changes. Typically, these are done to provide improved security to match modern security practices.
- ✓ **MINOR** versions contain enhancements but are considered passive updates.
- ✓ **PATCH** level should be perfectly compatible, forwards and backwards, with the possible exception of changes that fix bugs.

Getting Spring Security

pom.xml

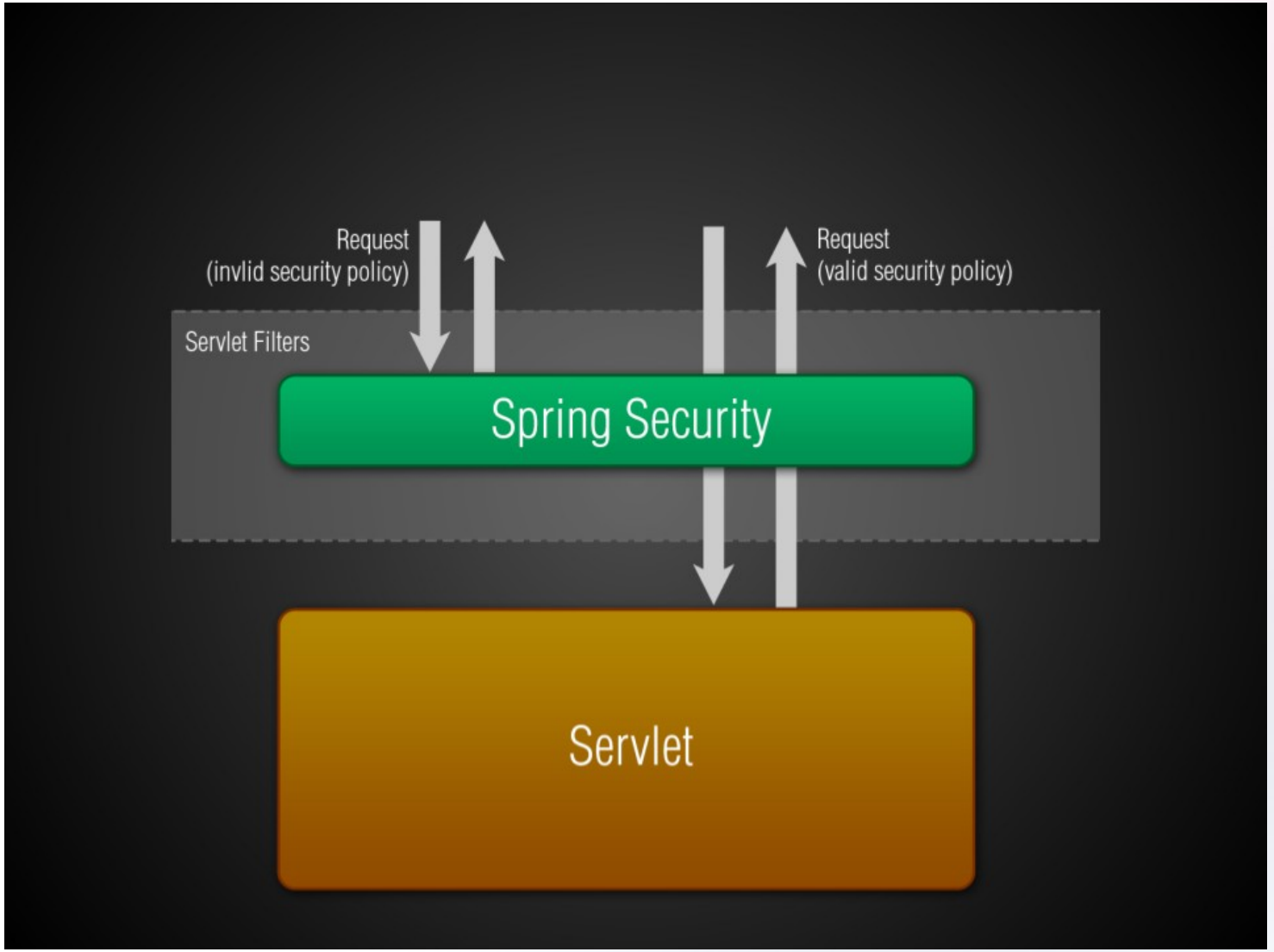
```xml
<dependencies>
    <!-- ... other dependency elements ... -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
</dependencies>
```

https://docs.spring.io/spring-security/reference/getting-spring-security.html

Features

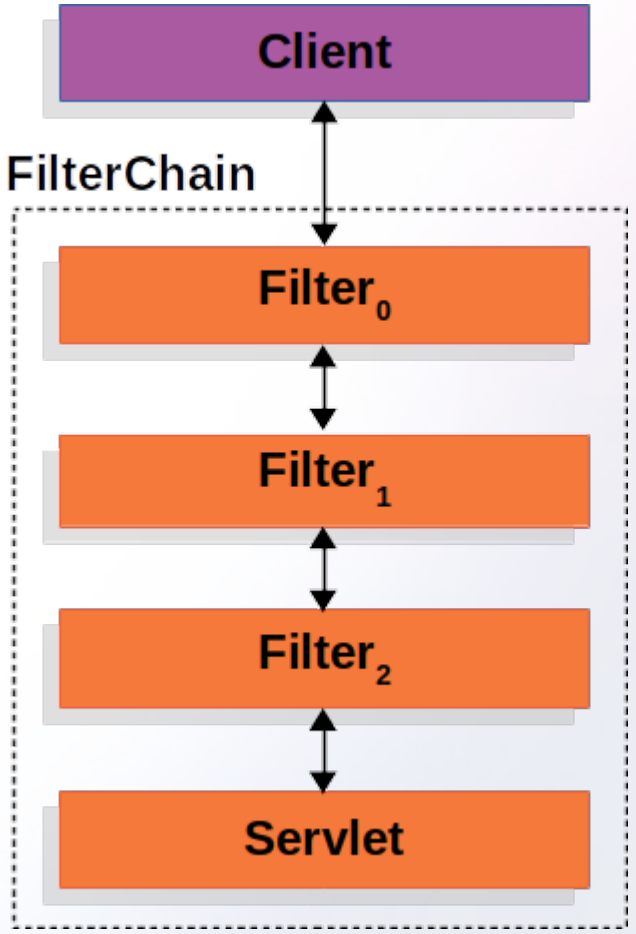
- Comprehensive and extensible support for both Authentication and Authorization
- Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
- Servlet API integration
- Optional integration with Spring Web MVC
- Much more…

Spring Security is a Servlet Filter



Spring Security Architecture

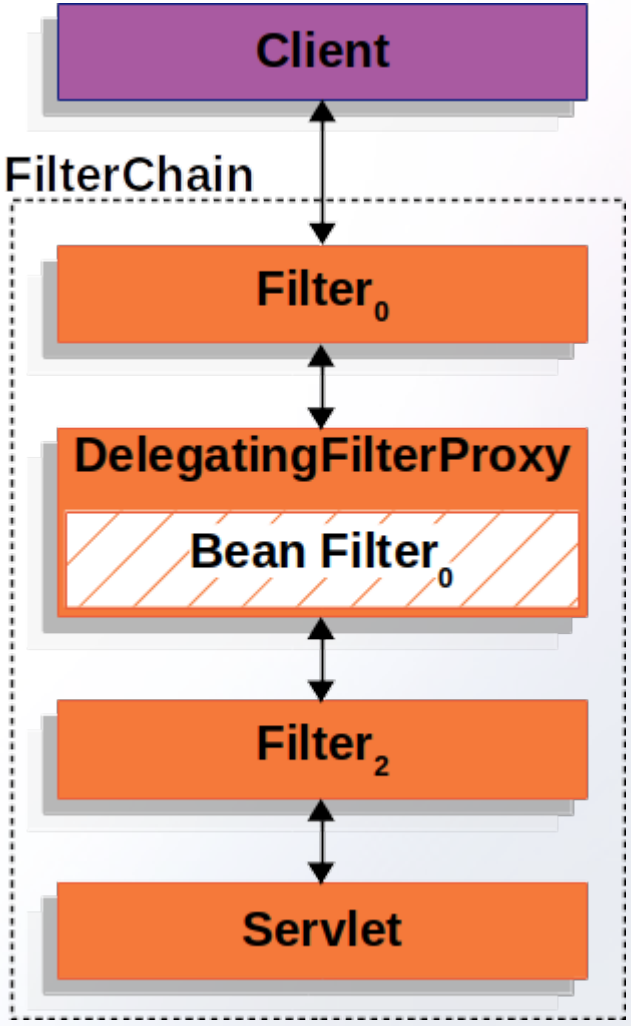
- https://docs.spring.io/spring-security/reference/servlet/architecture.html



sourcemind

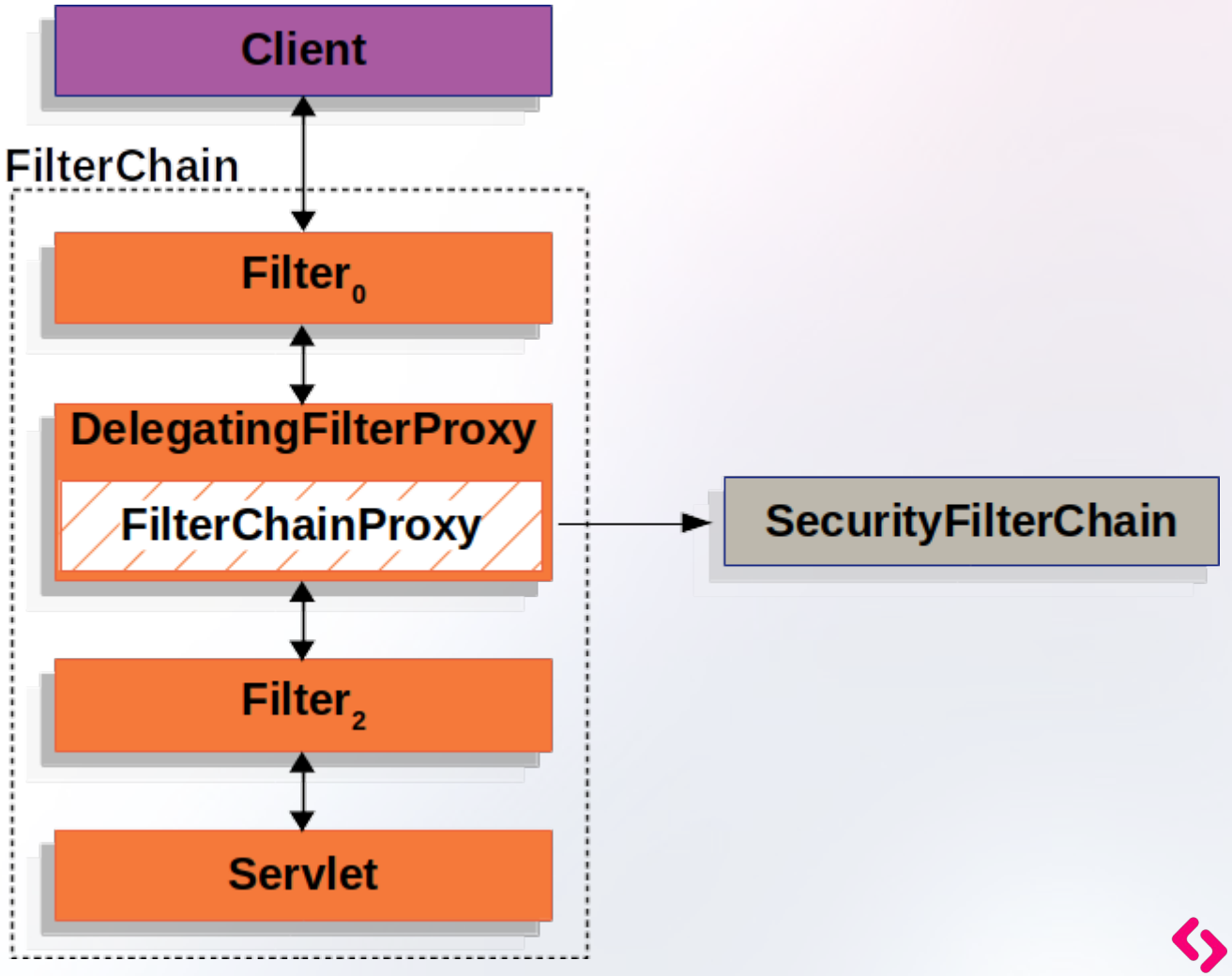
DelegatingFilterProxy

- Spring provides a Filter implementation named DelegatingFilterProxy that allows bridging between the Servlet container's lifecycle and Spring's ApplicationContext.



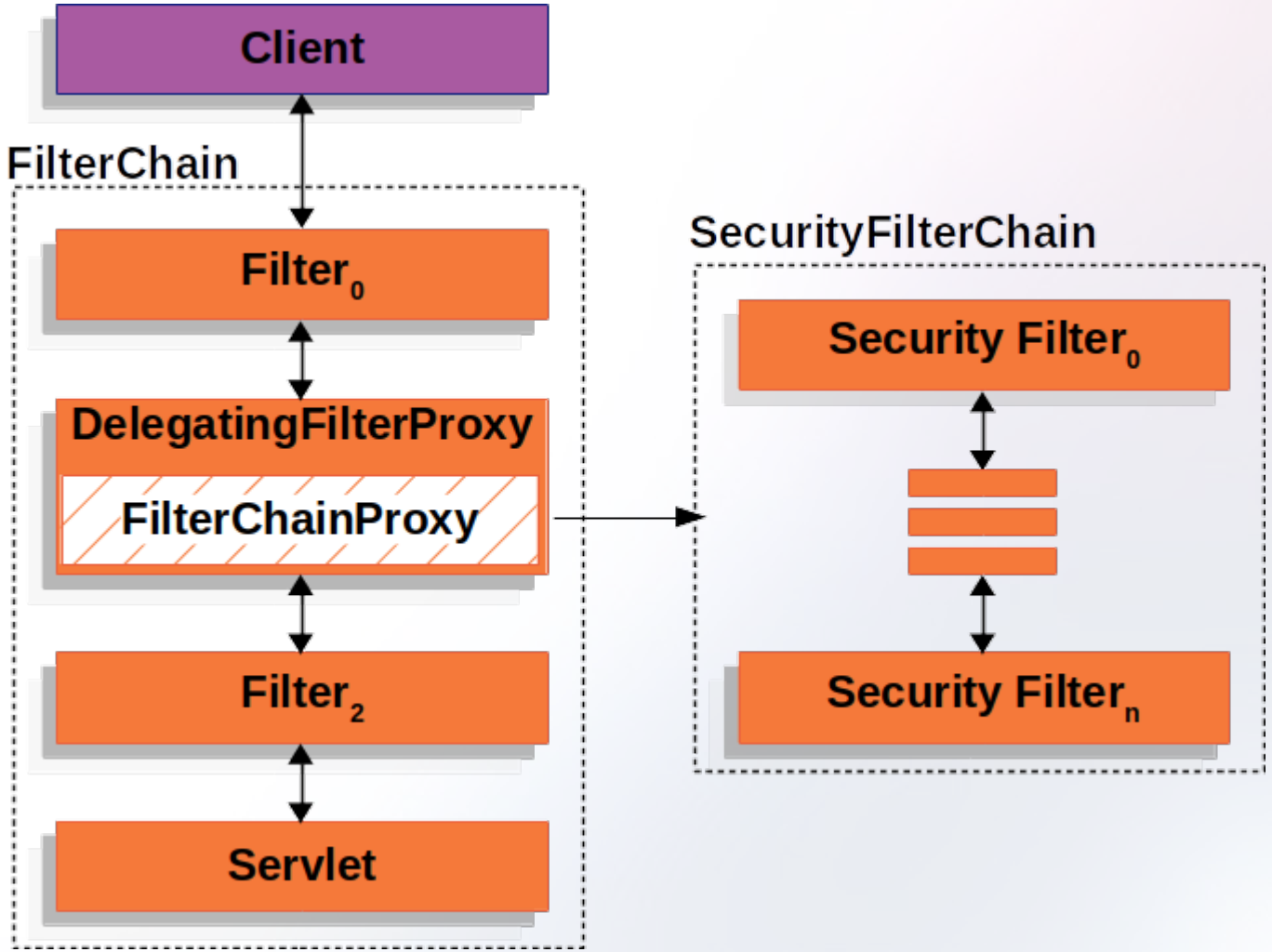
FilterChainProxy

- Spring Security's Servlet support is contained within FilterChainProxy.



SecurityFilterChain

- SecurityFilterChain is used by FilterChainProxy to determine which Spring Security Filter instances should be invoked for the current request.



Security Filters

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(Customizer.withDefaults())
            .authorizeHttpRequests(authorize -> authorize
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults())
            .formLogin(Customizer.withDefaults());
        return http.build();
    }

}
```

The above configuration will result in the following `Filter` ordering:

Filter	Added by
CsrfFilter	`HttpSecurity#csrf`
UsernamePasswordAuthenticationFilter	`HttpSecurity#formLogin`
BasicAuthenticationFilter	`HttpSecurity#httpBasic`
AuthorizationFilter	`HttpSecurity#authorizeHttpRequests`

Adding a Custom Filter to the Filter Chain

```
public class TenantFilter implements Filter {


    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;


        String tenantId = request.getHeader("X-Tenant-Id");
        boolean hasAccess = isUserAllowed(tenantId);
        if (hasAccess) {
            filterChain.doFilter(request, response);
            return;
        }
        throw new AccessDeniedException("Access denied");
    }
}
```

The sample code above does the following:

1. Get the tenant id from the request header.
2. Check if the current user has access to the tenant id.
3. If the user has access, then invoke the rest of the filters in the chain.
4. If the user does not have access, then throw an `AccessDeniedException`.

- Instead of implementing Filter, you can extend from OncePerRequestFilter which is a base class for filters that are only invoked once per request and provides a doFilterInternal method with the HttpServletRequest and HttpServletResponse parameters.

Adding a Custom Filter to the Filter Chain

```
@Bean
SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        // ...
        .addFilterBefore(new TenantFilter(), AuthorizationFilter.class);
    return http.build();
}
```

- Use HttpSecurity#addFilterBefore to add the TenantFilter before the AuthorizationFilter.

Security Filters Ordering

- ForceEagerSessionCreationFilter
- ChannelProcessingFilter
- WebAsyncManagerIntegrationFilter
- SecurityContextPersistenceFilter
- HeaderWriterFilter
- CorsFilter
- CsrfFilter
- LogoutFilter
- OAuth2AuthorizationRequestRedirectFilter
- Saml2WebSsoAuthenticationRequestFilter
- X509AuthenticationFilter
- AbstractPreAuthenticatedProcessingFilter
- CasAuthenticationFilter
- OAuth2LoginAuthenticationFilter
- Saml2WebSsoAuthenticationFilter
- UsernamePasswordAuthenticationFilter
- OpenIDAuthenticationFilter
- DefaultLoginPageGeneratingFilter
- DefaultLogoutPageGeneratingFilter
- ConcurrentSessionFilter
- DigestAuthenticationFilter
- BearerTokenAuthenticationFilter
- BasicAuthenticationFilter
- RequestCacheAwareFilter
- SecurityContextHolderAwareRequestFilter
- JaasApiIntegrationFilter
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter
- OAuth2AuthorizationCodeGrantFilter
- SessionManagementFilter
- ExceptionTranslationFilter
- FilterSecurityInterceptor
- SwitchUserFilter

Servlet Authentication Architecture

- https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html

End

sourcemind