# The Introduction of Spring Security

## What’s the Spring Security

Spring Security is the framework provided by Spring to support the process of authentication and authorization to protect endpoints.

Simply put, authentication is the process of verifying if username/password you provide when login are correct.

authorization is the process of if you have correct role permission to access to endpoints (resources).

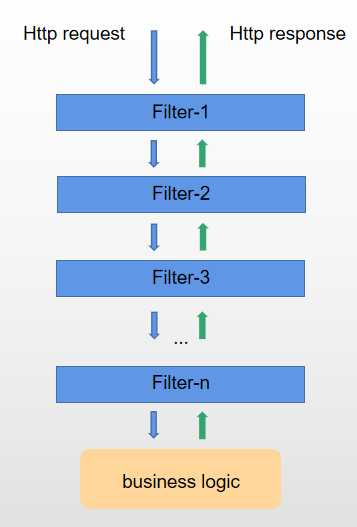
There are two important mechanisms to implement authentication and authorization. **Filter chain** and **Authentication Manager.**

They are two different mechanisms. However, they often work together to implement Spring Security

## Filter Chain

Filter chain is a set of filters in the form of chain. A http(s) request needs go through these filters (if available)before it arrives the business logic.

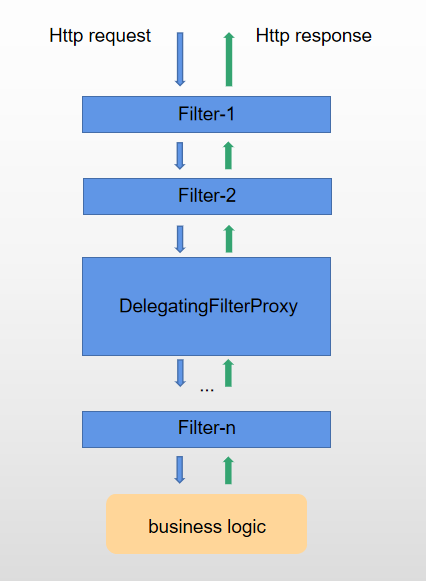
For the application without Spring Security, the filter chain may look like this:



Pic 1

Filter-x is a kind of servlet filter implementing **Filter** interface.

For the application with Spring Security, the filter chain looks like this:

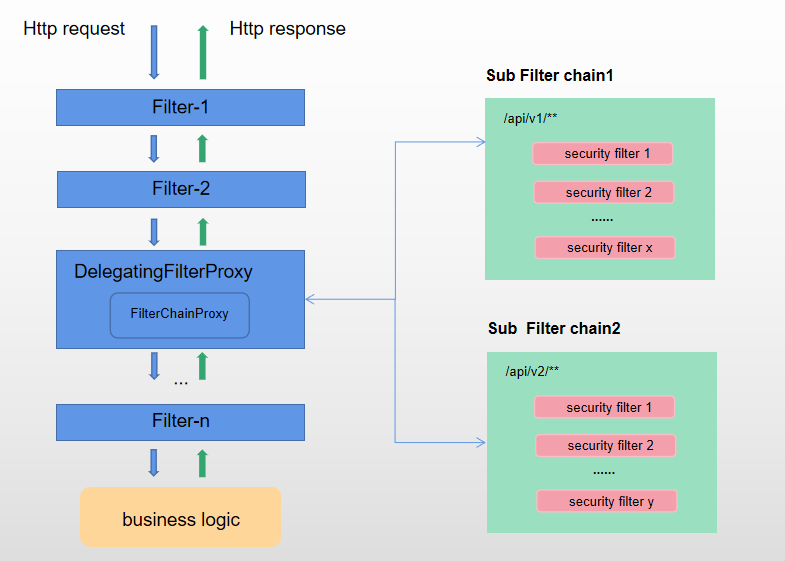


Pic 2

The filter chain in Pic 2 looks nothing special but a new filter **DelegatingFilterProxy** inserted compared with the previous one.

In fact, Once you enable spring security, DelegatingFilterProxy will be inserted automatically into filter chain(let’s call this filter chain as main filter chain). DelegatingFilterProxy is mainly responsible for the authentication and authorization.

Let’s take a look at the inside of DelegatingFilterProxy in detail.



Pic 3

*DelegatingFilterProxy* contains a *FilterChainProxy* which contains several sub filter chains. Each sub filter chain contains a set of filters. As shown in the Pic 3 for example, *FilterChainProxy* contains 2 sub filter chains: *sub filter chain1* and *sub filter chain2*. *sub filter chain1* contains a set of filters named *security filter1*, *security filter2*.. *security filter x*. *sub filter chain2* contains a set of filters named *security filter1*, *security filter2*.. *security filter y.*

But how do these filter chains and filters work together?

Let’s say you issue a request with url being “http://ip:8080/api/v1/\*\*” and request is accepted by all filters it goes through. Let’s see how it goes:

1. The request first arrives Filter-1, then it passes
2. The request arrives Filter-2, then it passes again
3. The request arrives *DelegatingFilterProxy, DelegatingFilterProxy* passes it down to *FilterChainProxy*
4. *FilterChainProxy* checks the url of this request and finds out it matches *sub filter chain1,* then *FilterChainProxy* passes the request down to *sub filter chain1*
5. *sub filter chain1* receives the request and passes it down to *security filter1*
6. The request passes *security filter1,* then goes down to *security filter2,* until *security filter x*
7. *sub filter chain1* is done once the request passes *security filter x. The request* continues the rest of filters behind *DelegatingFilterProxy* in the main filter chainuntil it arrives the business logic
8. The output of business logic is return in the order of Filter-n, ... *DelegatingFilterProxy,* Filter-2, Filter-1 (I am not sure if the return order is correct)

Yes. That is.

But how do we create such a complex filter chain graph? Do we need to create it from scratch? No. Spring Security has done the most of work: Such as automatically insert *DelegatingFilterProxy* into main filter chain*,* use *FilterChainProxy* to do url matching working. The only thing you need to do is to configure your own sub filter chain

## Configure sub filter chain

To create one sub filter chain, you just need to create a custom class extending WebSecurityConfigurerAdapter with @EnableWebSecurity. If you want to create 2 sub filter chains like the Pic3 above, simply create 2 custom classes extending WebSecurityConfigurerAdapter. Something like below:

@Configuration  
@EnableWebSecurity(debug = false)  
@Order(SecurityProperties.*BASIC\_AUTH\_ORDER* - 100)  
public class SubFilterChain1Config extends WebSecurityConfigurerAdapter {  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.antMatcher("/api/v1/\*\*");  
 }  
}

@Configuration  
@EnableWebSecurity(debug = false)  
@Order(SecurityProperties.*BASIC\_AUTH\_ORDER* - 99)  
public class SubFilterChain1Config extends WebSecurityConfigurerAdapter {  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.antMatcher("/api/v2/\*\*");  
 }  
}

## Authentication Manager

