

Spring Security is a powerful and highly customizable authentication and access-control framework.

If Spring Security is on the classpath, Spring Boot automatically secures all HTTP endpoints with “basic” authentication.

SecurityAutoConfiguration class containing the initial/default security configuration.

to disable auto configuration we need to exclude the SecurityAutoConfiguration class.

@SpringBootApplication(exclude = { SecurityAutoConfiguration.class })

or -spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration

The @EnableWebSecurity annotation is crucial if we disable the default security configuration.

Spring Security filters are registered with the lowest order and are the first filters invoked.

For some use cases, if you want to put your custom filter in front of them, you will need to add padding to their order.

spring.security.filter.order=10

AuthenticationProvider

Its interface exposes only two functions:

authenticate performs authentication with the request.

supports checks if this provider supports the indicated authentication type.

One important implementation of the interface DaoAuthenticationProvider which retrieves user details from a UserDetailsService.

Spring developers decided to extract it as a separate interface loadUserByUsername accepts username as a parameter and returns the user identity object.

JWT Token:\_\_

is an open Internet standard (RFC 7519) for securely transmitting trusted information between parties in a compact way.

A JWT token is divided into 3 parts namely – header, payload, and signature

[Header].[Payload].[Signature]

The Header of a JWT token contains the list cryptographic operations that are applied to the JWT. This can be the signing technique, metadata information about the content-type and so on.

{ "alg": "HS256", "typ": "JWT" }

The payload part of JWT contains the actual data to be transferred using the token. This part is also known as the “claims” part of the JWT token.

claims can be of three types – registered, public and private.

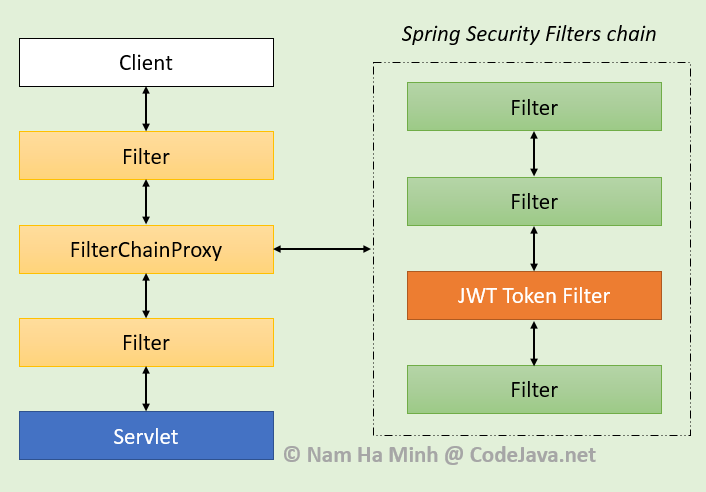
* The registered claims are the ones which are recommended but not mandatory claims such as iss(issuer), sub(subject), aud(audience) and others.
* Public claims are those that are defined by those using the JWTs.
* Private claims or custom claims are user-defined claims created for the purpose of sharing the information between the concerned parties.

{ "sub": "12345", "name": "Johnny Hill", "admin": false }

**Signature**− The signature part of the JWT is used for the verification that the message wasn’t changed along the way. If the tokens are signed with private key, it also verifies that the sender is who it says it is. It is created using the encoded header, encoded payload, a secret and the algorithm specified in the header.

HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

DelegatingFilterProxy





UsernamePasswordAuthenticationToken - authentication contains username and password in principal and credential.

BcryptPasswordEncoder in Spring Ecrypt uses salt mechanism.

**AuthenticationEntryPoint** is an interface implemented by *ExceptionTranslationFilter*, basically a filter which is the first point of entry for Spring Security. It is the entry point to check if a user is authenticated and logs the person in or throws exception (unauthorized). Usually the class can be used like that in simple applications but when using Spring security in REST, JWT etc one will have to extend it to provide better Spring Security filter chain management.

The AuthenticationEntryPoint is an interface that acts as a point of entry for authentication that determines if the client has included valid credentials when requesting for a resource. If not, an appropriate implementation of the interface is used to request credentials from the client.

**Order of Filter:**

ChannelProcessingFilter, because it might need to redirect to a different protocol

SecurityContextPersistenceFilter

ConcurrentSessionFilter

Authentication processing mechanisms - UsernamePasswordAuthenticationFilter,

SecurityContextHolderAwareRequestFilter

RememberMeAuthenticationFilter

AnonymousAuthenticationFilter

ExceptionTranslationFilter

FilterSecurityInterceptor

1. **WebAsyncManagerIntegrationFilter** Provides integration between the SecurityContext and Spring Web's WebAsyncManager.
2. **SecurityContextPersistenceFilter** This filter will only execute once per request, Populates the SecurityContextHolder with information obtained from the configured SecurityContextRepository prior to the request and stores it back in the repository once the request has completed and clearing the context holder.  
   *Request is checked for existing session. If new request, SecurityContext will be created else if request has session then existing security-context will be obtained from respository*.
3. **HeaderWriterFilter** Filter implementation to add headers to the current response. It is basically used for enable browser security. Like X-Frame-Options, X-XSS-Protection and X-Content-Type-Options.
4. **LogoutFilter** If request url is /logout(for default configuration) or if request url matches RequestMatcher configured in LogoutConfigurer then

* clears security context.
* invalidates the session
* deletes all the cookies with cookie names configured in LogoutConfigurer
* Redirects to default logout success url / or logout success url configured or invokes logoutSuccessHandler configured.

1. **UsernamePasswordAuthenticationFilter**

* For any request url other than *loginProcessingUrl* this filter will not process further, but filter chain just continues.
* If requested URL is matches(must be HTTP POST) default /login or matches .loginProcessingUrl() configured in FormLoginConfigurer then UsernamePasswordAuthenticationFilter attempts authentication.
* default login form parameters are username and password, can be overridden by usernameParameter(String), passwordParameter(String).
* setting .loginPage() [overrides defaults](https://i.stack.imgur.com/apOKt.png)
* While attempting authentication

> an [Authentication](https://i.stack.imgur.com/0TQS8.png) object(UsernamePasswordAuthenticationToken or any implementation of Authentication in case of your custom auth filter) is created.

> and authenticationManager.authenticate(authToken) will be invoked

> Note that we can configure any number of AuthenticationProvider authenticate method tries all auth providers and checks any of the auth provider ***supports*** authToken/authentication object, supporting auth provider will be used for authenticating. and returns Authentication object in case of successful authentication else throws AuthenticationException.

* If authentication success session will be created and authenticationSuccessHandler will be invoked which redirects to the target url configured(default is /)
* If authentication failed user becomes un-authenticated user and chain continues.

1. **SecurityContextHolderAwareRequestFilter**, if you are using it to install a Spring Security aware HttpServletRequestWrapper into your servlet container
2. **AnonymousAuthenticationFilter** Detects if there is no Authentication object in the SecurityContextHolder, if no authentication object found, creates Authentication object (AnonymousAuthenticationToken) with granted authority ROLE\_ANONYMOUS. Here AnonymousAuthenticationToken facilitates identifying un-authenticated users subsequent requests.
3. **ExceptionTranslationFilter**, to catch any Spring Security exceptions so that either an HTTP error response can be returned or an appropriate AuthenticationEntryPoint can be launched
4. **FilterSecurityInterceptor**  
   There will be FilterSecurityInterceptor which comes almost last in the filter chain which gets Authentication object from SecurityContext and gets granted authorities list(roles granted) and it will make a decision whether to allow this request to reach the requested resource or not, decision is made by matching with the allowed AntMatchers configured in HttpSecurityConfiguration.