# Spring Security.

**Authentication and Authorization**

A very short spring security descrption in security-xml file looks like this

<http auto-config="true">

<intercept-url pattern="/users/input\*" access="ROLE\_USER" />

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name="susil" password="12345" authorities="ROLE\_USER" />

</user-service>

</authentication-provider>

</authentication-manager>

Another kind

<http auto-config="true">

<intercept-url pattern="/messageList\*"

access="ROLE\_USER,ROLE\_ANONYMOUS" />

<intercept-url pattern="/messagePost\*" access="ROLE\_USER" />

<intercept-url pattern="/messageDelete\*" access="ROLE\_ADMIN" />

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name="admin" password="secret"

authorities="ROLE\_ADMIN,ROLE\_USER" />

<user name="user1" password="1111" authorities="ROLE\_USER" />

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

HTTP Basic Authentication

The HTTP Basic authentication support can be configured via the <http-basic> element. When HTTP Basic

authentication is required, a browser will typically display a login dialog or a specific login page for users to log in.

<http>

...

<http-basic />

</http>

Form-Based Login

The form-based login service will render a web page that contains a login form for users to input their login details

and process the login form submission. It’s configured via the <form-login> element:

<http>

...

<form-login />

</http>

By default, Spring Security automatically creates a login page and maps it to the URL /spring\_security\_login.

So, you can add a link to your application.

* *AuthenticationManager* - handles authentication requests from other parts of the framework.

*AuthenticationProvider*s - mechanisms against which the authentication manager authenticates users. The namespace provides supports for

several standard options and also a means of adding custom beans declared using a traditional syntax.

* *UserDetailsService* - closely related to authentication providers, but often also required by other beans.

## How to implement Spring security for a web application.

## **web.xml Configuration**

The first thing you need to do is add the following filter declaration to your web.xml file:

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

DelegatingFilterProxy is a spring security framework class which propagates control to a filter implementation which is defined as a Spring bean in your application context. In this case, the bean is named “springSecurityFilterChain”, which is an internal infrastructure bean created by spring web security.

Minimal Http Configuration

<http auto-config='true'>

<intercept-url pattern="/\*\*" access="ROLE\_USER" />

</http>

Adding some users as

<authentication-manager>

<authentication-provider>

<user-service>

<user name="john" password="johny" authorities="ROLE\_USER, ROLE\_ADMIN" />

<user name="john\_ad" password="johnadmin" authorities="ROLE\_USER" />

</user-service>

</authentication-provider>

</authentication-manager>

So here we have got two users with role mentioned.

User Authentication Providers.

Own authentication provider.

Create a class and implements UserDetailsService and use it as

<authentication-manager>

<authentication-provider user-service-ref='customAuthenicator'/>

</authentication-manager>

entery-point-ref:

================

If you aren't using form login or basic authentication through the namespace, you may want to define an authentication filter and entry point as

<http-basic entry-point-ref="clientAuthenticationEntryPoint" /> where it will authenticate or take the burden of authentication.

In Oauth2 I'v used this where I'm passing userid and password as part of url or json packet where user should not be asked with a login page.

## FILTER STACK IN SPRING SECURITY

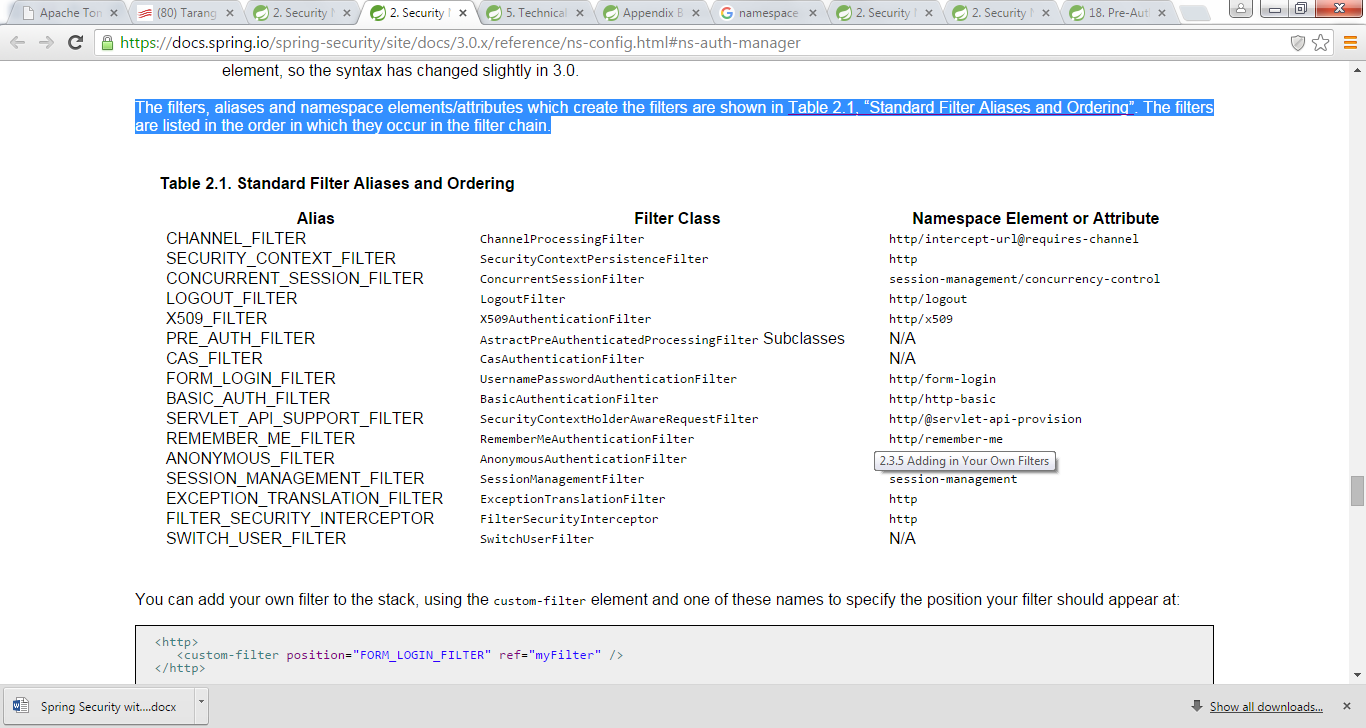
Spring security framework maintains a chain of filters in order to apply its services. You may want to add your own filters to the stack at particular locations or use a Spring Security filter for which there isn't currently a namespace configuration option

Or you might want to use a customized version of a standard namespace filter, such as the UsernamePasswordAuthenticationFilter which is created by the <form-login> element, taking advantage of some of the extra configuration options which are available by using the bean explicitly. How can you do this with namespace configuration, since the filter chain is not directly exposed?

You can use after or before attribute in <custom-filiter> tag to specify what filter you want to attached and at what location.

The order of the filters is always strictly enforced when using the namespace. When the application context is being created, the filter beans are sorted by the namespace handling code and the standard Spring Security filters each have an alias in the namespace and a well-known position.

The filters, aliases and namespace elements/attributes which create the filters are shown in below table. The filters are listed in the order in which they occur in the filter chain.



You can add your own filter to the stack, using the custom-filter element and one of these names to specify the position your filter should appear at:

<http>

<custom-filter position="FORM\_LOGIN\_FILTER" ref="myFilter" />

</http>

<beans:bean id="myFilter" class="com.mycompany.MySpecialAuthenticationFilter"/>

You can also use the after or before attributes if you want your filter to be inserted before or after another filter in the stack. The names "FIRST" and "LAST" can be used with the position attribute to indicate that you want your filter to appear before or after the entire stack, respectively.

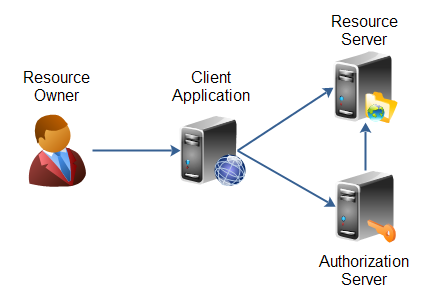
After: it says after which filter you want to inject your filter..

Setting a Custom AuthenticationEntryPoint

If you aren't using form login, OpenID or basic authentication through the namespace, you may want to define an authentication filter and entry point using a traditional bean syntax and link them into the namespace, as we've just seen. The corresponding AuthenticationEntryPoint can be set using theentry-point-ref attribute on the <http> element.

# Oauth2.0

OAUTH.



With Authorization Server (Oauth implementation) an authenticated user can be authorized to access **resource server**.

Here I’v implemented the whole flow as two-step process

* Spring Security for authentication. (Spring security is req for Spring Oauth)
* Spring Oauth2.0 for authorization.

Letz say I’v a web service that I want to authorize to it’s valid users.

Assuming the web service could be accessed as

<http://localhost:8080/mywebservice>

Steps for integrating Oauth2.0

* Add spring security dependency to web service and configure it.
* Add spring Oauth2.0 dependency to web service and configure it.

What Configure mean by: While configuring we need to specify which area of web service (part of url) needed to be exposed to what users, map the required classes that will take the burden of authentication, what kind of authentication require (authorization code, implicit, resource owner password credential, client credential).

Steps for accessing a resource:

1. For accessing any resource the client application need to register with the Oauth Server to get Client id and Client secret key.
2. Client application also need have user id and password for the user whose resource we want to access.
3. The client application should obtain a token from the Oauth server

<http://localhost:8080/mywebservice>/oauth/token? username=user1&password=user1&client\_id=client1&client\_secret=client1&grant\_type=password

(we can sent this data as part of post).

For the above steps the Oauth server will first test the client id and secret for confirming a valid client application or not , then it will go on testing user id and password.

And for valid data it responds with json data

{"access\_token":"aa45e9ac-83a1-4977-898b-43d1eb9933b8","token\_type":"bearer","refresh\_token":"e8208a2c-4b25-425e-a360-23a1b68de315","expires\_in":298825}

Access\_token: should be accompanied while requesting any resource.

Bearer : says anyone who got the token can access resource.

Refresh\_token: refresh\_token is used for getting a fresh token when the token expires.

Expires\_in: specified time after which the token become invalid.

4)

Once the client application got the token he can access resource for the user, using the given token

As   
<http://localhost:8080/mywebservice/profile_page/?access_token=b6a50b54-0cd1-476f-8539-b3b5ffd8f406>

5) Client application need to refresh the token after the expiration period.

**OAUTH FRAMEWORK [LOGIN SCENARIO]**

Web Se rvices / Cloud Server

Oauth Server

FILTER

CLIENT/MOBLE APP

Client accessing login URL with user credential.

Filter intercepts the client requests and talk to auth server for validation and token generation with client credential

Oauth Server provides response to filter with a token for valid user else with an error message

Oauth Server will access cloud for validating user with client credential credential.

Cloud respond to oauth server for user validation

If token is generated by Oauth Server, filter will create an entry in LoginMap with the token.

Filter respond to client with a token for valid user else respond with an error message.

Login Flow (Coincide below steps to above labelled arrows)

1. Client application accessing login service with server url (<http://localhost:8080/mobileApp/rest/user/login>) with client credential.
2. Filter will intercept the client request and talk to oauth with client credential for getting a token (token will only be fetched for an authenticated client)
3. Oauth server will access cloud server for validating the client credential.
4. Cloud server will respond to Oauth server with validation status.(true for valid user else false).
5. Upon getting the response, Oauth server will respond to filter. For a valid user it respond with a token.
6. If a token is returned from OauthServer (in case of a valid user).

* Filter will create an entry in the LoginMap ( where it keeps the token as key and UserPojo as an object).

For an invalid user ( when OauthServer responded with validation error message).

1. For a valid user Filter will respond to client with a token else Error message will be returned.

OauthServer: In the above framework, Oauth Server works as an independent module, which generates an unique token with a token expiration period.

Filter: Filter intercepts each of the client request before it hits to cloud server, so that validation can be accomplished in Filter.

LoginMap : It’s a map which contains token as key and UserPojo as value pair. When the user trying to access any service with a valid token , cloud server be identified the user referring the LoginMap.

**[Post login scenario]**

Web Servcie / Cloud Server

Oauth Server

FILTER

CLIENT/MOBLE APP

Client accessing a service URL with a token

Filter access the Oauth server to confirm on token expiration.

Oatuh server respond to filter with appropriate message. For an expired token filter removes the token from LoginMap and ask the user to login again.

For a valid token filter allows the request to access cloud server

Here the response from the cloud received by the filter.

If client accessing with an invalid or expired token filter ask the client to login, else hands over the client with cloud response.

Post login scenario.

1. User accessing any cloud service url with a token.
2. Filter will intercept the user request, fetch the token and talk to Oauth Server for authenticating the token.

* If the token is authenticated , filter will allow access to cloud
* For an invalid token it will respond with a validation message
* For an expired token it will, remove the entry from the LoginMap and ask the user to login again.

1. Filter receives the response from the cloud server and for an expired token it removes the entry from LoingMap.
2. For a valid token, Filter allows the request to access the cloud server.
3. Filter receives the client response.
4. Filter respond to client with cloud response.

**Logout Scenario.**

If the user access the LogOut Url with the valid token, Filter will remove the entry in LoginMap.

# Configuration for oauth

<http pattern=*"/oauth/token"* create-session=*"stateless"*

authentication-manager-ref=*"authenticationManager"*

xmlns=*"http://www.springframework.org/schema/security"* >

<intercept-url pattern=*"/oauth/token"* access=*"IS\_AUTHENTICATED\_FULLY"* />

<anonymous enabled=*"false"* />

<http-basic entry-point-ref=*"clientAuthenticationEntryPoint"* />

<custom-filter ref=*"clientCredentialsTokenEndpointFilter"* before=*"BASIC\_AUTH\_FILTER"* />

<access-denied-handler ref=*"oauthAccessDeniedHandler"* />

</http>

* authentication-manager-ref="authenticationManager" : it is used for authenticating user id and password.
* In the above markup <http-basic entry-point-ref=*"clientAuthenticationEntryPoint"* />

Is used for validating user id and password without showing up a login screen.

* Custom filter is injected that will take care about client id and secret and it will be called before Basic\_auth\_filter, Basic\_auth\_filter will take care for validating user id and password based on authentication manager.

If above succeeds it will lend you a token.

<http pattern=*"/resources/\*\*"* create-session=*"never"*

entry-point-ref=*"oauthAuthenticationEntryPoint"*

xmlns=*"http://www.springframework.org/schema/security"*>

<anonymous enabled=*"false"* />

<intercept-url pattern=*"/resources/\*\*"* access=*"IS\_AUTHENTICATED\_FULLY"* />

<intercept-url pattern=*"/resources/MyResource/createInfo\*"* />

<custom-filter ref=*"resourceServerFilter"* before=*"PRE\_AUTH\_FILTER"* />

<access-denied-handler ref=*"oauthAccessDeniedHandler"* />

</http>

Once the user is authenticated and got the token, he will try to access the some urls with access\_token, that access token needs to be validated with urls.

For that we need to provide another custom filter that will check the access\_token is vaild or not….

resourceServerFilter is a class given by Oauth2.0 spring.

<oauth:resource-server id=*"resourceServerFilter"*

resource-id=*"springsec"* token-services-ref=*"tokenServices"* />

<bean id=*"tokenStore"*

class=*"org.springframework.security.oauth2.provider.token.InMemoryTokenStore"* />

<bean id=*"tokenServices"*

class=*"org.springframework.security.oauth2.provider.token.DefaultTokenServices"*>

<property name=*"tokenStore"* ref=*"tokenStore"* />

<property name=*"supportRefreshToken"* value=*"true"* />

<property name=*"accessTokenValiditySeconds"* value=*"600"*></property>

<property name=*"clientDetailsService"* ref=*"clientDetails"* />

</bean>

How client id and secrets is validated with custom class

<authentication-manager alias=*"authenticationManager"*

xmlns=*"http://www.springframework.org/schema/security"*>

<authentication-provider user-service-ref=*"clientDetailsUserService"* />

</authentication-manager>

<bean id=*"clientDetailsUserService"*

class=*"org.springframework.security.oauth2.provider.client.ClientDetailsUserDetailsService"*>

<constructor-arg ref=*"clientDetails"* />

</bean>

<bean id="clientDetails" class="org.omnypay.oauth.ClientDetailsServiceImpl"/>

How user id and password is validated

<authentication-manager id="userAuthenticationManager"

xmlns="http://www.springframework.org/schema/security">

<authentication-provider ref="customUserAuthenticationProvider">

</authentication-provider>

</authentication-manager>

Note: customUserAuthenticationProvider is the bean for which the class is

<bean id="customUserAuthenticationProvider"

class="org.omnypay.oauth.CustomUserAuthenticationProvider">

but you will find above bin is commented but autowired.

# Notes on java classes

## Class: ClientDetailsServiceImpl //implements clientDetailsService

When oauth server is hit by a url as to get a token

<http://localhost:9095/Oauth/oauth/token?username=tarang&password=oauth&client_id=client1&client_secret=client1&grant_type=password>

Ouath server need to validate two thing ,

1. one is client id and secret
2. user id and password of the user.

ClientDetailsServiceImpl used by oauth framework for validating client id and client secret.

Override the method

public ClientDetails loadClientByClientId(String clientId)throws OAuth2Exception {

//load client secrete based on passed client id and initialize with authorization grant type and return , ouath framework

}

## Class: CustomUserAuthenticationProvider //implement AuthenticationProvider

This class is used by oauth framework to validate the user credential, user id and password.

Override the method

public Authentication authenticate(Authentication authentication){

//access userid and password from authentication object and return

An object of AbstractAuthenticationToken else raise

throw new BadCredentialsException("Bad User Credentials.");

}

## Class: CustomUserPasswordAuthenticationToken // implements AbstractAuthenticationToken

This class represet user id and password.

## Class: UserAuthenticationServiceImpl

class UserAuthenticationServiceImpl , this class is used for validating user id nad password with the cloud web service.