OAuth2 Provider support for the Spring Security plugin.

Spring Security OAuth2 Plugin - Reference Documentation

Authors:

Version: 2.0-RC3

Table of Contents

1 Introduction to the Spring Security OAuth2 Plugin

The OAuth2 plugin adds OAuth 2.0 support to a Grails application that uses Spring Security. It depends or

Under the covers, Spring Security OAuth version 2.0.2.RELEASE is used by the plugin to provide OAuth ensure proper integration with the underlying library.

This plugin provides support for Grails domain classes necessary for providing OAuth 2.0 authorization. T resources is controlled by a combination of Spring Security Core's methods, i.e. request maps, annotations,

1.1 Change Log

- 2.0-RC3
  - Upgrade to Spring OAuth 2.0.6.RELEASE (issue #63)
  - Fix problems with updating access tokens (issues #49, #50, and #68)
  - Add TravisCI build
  - Ensure Set-Cookie header is not set in response
  - Fix handling of scope parameter (issue #64)
- 2.0-RC2
  - Resolves session vulnerability (issue #42)
  - Upgrade to Spring Security OAuth2 2.0.4.RELEASE
  - Supports authorization auto-approval
  - Minor tweaks to domain models
- 2.0-RC1
  - Complete overhaul of the plugin
  - Requires/supports Spring Security Core 2.0-RC4
  - Uses Spring Security OAuth2 2.0.2.RELEASE
- 1.0.5.2
  - Fix #13 - Make clientSecret optional in client configuration structure
- 1.0.5.1
  - Merge pull request #21 (Burt's cleanup)
  - Use log wrapper instead of log4j
  - Depends on Grails 2.0 or greater (consistent with core plugin)
- 1.0.5
  - Initial release of plugin compatible with spring security core 2.0-RC2

2 Getting Started

The following assumes that the Spring Security Core plugin has been installed and its required domain clas

2.1 Install Plugin

Install the OAuth2 plugin by adding a dependency in `grails-app/conf/BuildConfig.groovy`:

```
plugins {
    compile ":spring-security-oauth2-provider:2.0-RC3"
}
```

This has a dependency on the Spring Security Core plugin, which will be installed if necessary.

2.2 Create Domain Classes

Run the s2-init-oauth2-provider script to generate the required domain classes.

2.3 Secure Authorization and Token Endpoints

Update the Core plugin's rules for the authorization and token endpoints so they are protected by Spring S
`grails-app/conf/Config.groovy`:

```
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
        '/oauth/authorize.dispatch':      ["isFullyAuthenticated() and (request.g
        '/oauth/token.dispatch':          ["isFullyAuthenticated() and request.ge
        ...
```

The endpoints are standard Spring MVC controllers in the underlying Spring Security OAuth2 implementa

The additional restrictions on the allowed HTTP methods are to ensure compliance with the OAuth 2.0 spe

2.4 Add Client Provider

Next you will need to add the `clientCredentialsAuthenticationProvider` to the list of prov

```
grails.plugin.springsecurity.providerNames = [
        'clientCredentialsAuthenticationProvider',
        'daoAuthenticationProvider',
        'anonymousAuthenticationProvider',
        'rememberMeAuthenticationProvider'
]
```

The order is important. The `clientCredentialsAuthenticationProvider` **must** occur first Credentials grant.

2.5 Exclude client_secret From Logs

Update the params exclusion list in `grails-app/conf/Config.groovy` so client secrets are not log

```
grails.exceptionresolver.params.exclude = ['password', 'client_secret']
```

2.6 (Optional) Customize Error and Confirm Access Views

When the plugin is installed, two views are copied for the error and confirm access pages. They are located

2.7 Client Registration

At this point your application is a proper OAuth 2.0 provider. You can now register clients in what `grails-app/conf/Bootstrap.groovy` as follows:

```
def init = { servletContext ->
        new Client(
                clientId: 'my-client',
                authorizedGrantTypes: ['authorization_code', 'refresh_token', 'im
                authorities: ['ROLE_CLIENT'],
                scopes: ['read', 'write'],
                redirectUris: ['http://myredirect.com']
        ).save(flush: true)
    }
```

2.8 Controlling Access to Resources

Access to resources is controlled by the Spring Security Core plugin's access control mechanisms. Additio Spring library. Refer to the methods in OAuth2SecurityExpressionMethods for what is available in the plu

Using SPeL is the only tested and confirmed way to enforce OAuth 2.0 specific restrictions on resource ac

The following controller illustrates the use of OAuth 2.0 SPeL:

```
class SecuredOAuth2ResourcesController {

@Secured(["#oauth2.clientHasRole('ROLE_CLIENT')"])
    def clientRoleExpression() {
        render "client role expression"
    }

@Secured(["ROLE_CLIENT"])
    def clientRole() {
        render "client role"
    }

@Secured(["#oauth2.clientHasAnyRole('ROLE_CLIENT', 'ROLE_TRUSTED_CLIENT')"])
    def clientHasAnyRole() {
        render "client has any role"
    }

@Secured(["#oauth2.isClient()"])
    def client() {
        render "is client"
    }

@Secured(["#oauth2.isUser()"])
    def user() {
        render "is user"
    }

@Secured(["#oauth2.denyOAuthClient()"])
    def denyClient() {
        render "no client can see"
    }

@Secured(["permitAll"])
    def anyone() {
        render "anyone can see"
    }

def nobody() {
        render "nobody can see"
    }

@Secured(["#oauth2.clientHasRole('ROLE_TRUSTED_CLIENT') and #oauth2.isClient() an
    def trustedClient() {
        render "trusted client"
    }

@Secured(["hasRole('ROLE_USER') and #oauth2.isUser() and #oauth2.hasScope('trust'
    def trustedUser() {
        render "trusted user"
    }

@Secured(["hasRole('ROLE_USER') or #oauth2.hasScope('read')"])
    def userRoleOrReadScope() {
        render "user role or read scope"
    }
}
```

The filter chains must be configured to ensure stateless access to the token endpoint and any OAuth 2.0 res

```
grails.plugin.springsecurity.filterChain.chainMap = [
        '/oauth/token':
'JOINED_FILTERS,-oauth2ProviderFilter,-securityContextPersistenceFilter,-logoutFi
        '/securedOAuth2Resources/**':
'JOINED_FILTERS,-securityContextPersistenceFilter,-logoutFilter,-rememberMeAuthen
        '/**':
'JOINED_FILTERS,-statelessSecurityContextPersistenceFilter,-oauth2ProviderFilter,
]
```

Please consult the section on Filter Chain Configuration for more information.

2.9 Trouble Shooting

If you encounter a `NullPointerException` while using the OAuth2 plugin, you might have run in Cache Plugin. However, the latest version at the time of this writing (`1.1.6`) seems to have fixed this uninstall it.

3 Example Flows

The following examples assume you have followed the steps outlined in the Getting Started section for a
contains the following:

```
def init = { servletContext ->

Role roleUser = new Role(authority: 'ROLE_USER').save(flush: true)

User user = new User(
        username: 'my-user',
        password: 'my-password',
        enabled: true,
        accountExpired: false,
        accountLocked: false,
        passwordExpired: false
    ).save(flush: true)

UserRole.create(user, roleUser, true)

new Client(
        clientId: 'my-client',
        authorizedGrantTypes: ['authorization_code', 'refresh_token', 'implicit',
        authorities: ['ROLE_CLIENT'],
        scopes: ['read', 'write'],
        redirectUris: ['http://myredirect.com']
    ).save(flush: true)
}
```

After retrieving an `access_token` via one of the flows, you must include this in the `Authorization`

For example, if you receive `7b9a989e-3702-4621-a631-fbd1a996fc94` as the `acce`
`7b9a989e-3702-4621-a631-fbd1a996fc94` when requesting a protected resource.

3.1 Authorization Code Grant

The authorization code grant flow is initiated by directing your browser to the authorization endpoint:

```
http://localhost:8080/oauth2-test/oauth/authorize?response_type=code&client_id=my
```

You will be redirected to the login page. After signing in, you will be prompted to confirm the request. Do

```
http://myredirect.com/?code=139R59
```

The authorization code included in the query can be exchanged for an access token via the token endpoint:

```
http://localhost:8080/oauth2-test/oauth/token?grant_type=authorization_code&code=
```

You'll receive the `access_token` in the response:

```
{
    "access_token": "a1ce2915-8d79-4961-8abb-2c6f0fdb4aba",
    "token_type": "bearer",
    "refresh_token": "6540222d-0fb9-4b01-8d45-7be2bdfb68f9",
    "expires_in": 43199,
    "scope": "read"
}
```

3.2 Implicit Grant

The implicit grant is similar to the authorization code grant and can be initiated by directing your browser t

```
http://localhost:8080/oauth2-test/oauth/authorize?response_type=token&client_id=m
```

Upon confirmation, your browser will be redirected to the following URL:

```
http://myredirect.com/#access_token=4e22ad4f-08ae-49dc-befb-2c9821af04d1&token_ty
```

The `access_token` can be extracted from the URL fragment.

3.3 Resource Owner Password Credentials Grant

The resource owner password grant is performed by requesting an access token from the token endpoint:

```
http://localhost:8080/oauth2-test/oauth/token?grant_type=password&client_id=my-cl
```

The `access_token` is included in the response:

```
{
    "access_token": "1d49fc35-2af6-477e-8fd4-ab0353a4a76f",
    "token_type": "bearer",
    "refresh_token": "4996ba33-be3f-4555-b3e3-0b094a4e60c0",
    "expires_in": 43199,
    "scope": "read"
}
```

3.4 Client Credentials Grant

The client credentials grant is performed by authenticating the client via the token endpoint:

```
http://localhost:8080/oauth2-test/oauth/token?grant_type=client_credentials&clien
```

The `access_token` can be extracted from the response:

```
{
    "access_token": "7b9a989e-3702-4621-a631-fbd1a996fc94",
    "token_type": "bearer",
    "expires_in": 43199,
    "scope": "read"
}
```

3.5 Refresh Token Grant

The refresh token grant is performed by exchanging a refresh token received during a previous authorizatic

```
http://localhost:8080/oauth2-test/oauth/token?grant_type=refresh_token&refresh_to
```

The above assumes that `269afd46-0b41-45c2-a920-7d5af8a38d56` is the value of the refresh to

The `access_token` is included in the response:

```
{
    "access_token": "a3da52c7-4bd2-4d42-a58d-efa64b4de453",
    "token_type": "bearer",
    "refresh_token": "6396c283-47ff-41d2-b887-39bde6af5f1e",
    "expires_in": 43199,
    "scope": "read"
}
```

4 Required Domain Classes

The plugin uses regular Grails domain classes backed by GORM. There are four required domain classes r

The s2-init-oauth2-provider script will create the domain classes for you in a specified package and updat the generated classes to fit your needs. If you change the default property names, you will need to update g on domain class properties for more information.

> ⚠ The `maxSize` constraints in the generated domain classes have been set to reasonable def usernames (email addresses for example), or have many authorities attached to a single user.

The below discussion assumes the s2-init-oauth2-provider script has been run with `com.your` `AuthorizationCode` as the names of your domain classes.

4.1 Client Class

Information from the Grails client domain class will be extracted to create a `ClientDetails` instance fo

The generated class will look like this:

```
package com.yourapp

class Client {

private static final String NO_CLIENT_SECRET = ''

transient springSecurityService

String clientId
    String clientSecret

Integer accessTokenValiditySeconds
    Integer refreshTokenValiditySeconds

Map<String, Object> additionalInformation

static hasMany = [
            authorities: String,
            authorizedGrantTypes: String,
            resourceIds: String,
            scopes: String,
            autoApproveScopes: String,
            redirectUris: String
    ]

static transients = ['springSecurityService']

static constraints = {
        clientId blank: false, unique: true
        clientSecret nullable: true

accessTokenValiditySeconds nullable: true
        refreshTokenValiditySeconds nullable: true

authorities nullable: true
        authorizedGrantTypes nullable: true

resourceIds nullable: true

scopes nullable: true
        autoApproveScopes nullable: true

redirectUris nullable: true
        additionalInformation nullable: true
    }

def beforeInsert() {
        encodeClientSecret()
    }

def beforeUpdate() {
        if(isDirty('clientSecret')) {
            encodeClientSecret()
        }
    }

protected void encodeClientSecret() {
        clientSecret = clientSecret ?: NO_CLIENT_SECRET
        clientSecret = springSecurityService?.passwordEncoder ? springSecuritySer
    }
}
```

The client secret is encoded using the same strategy that is configured by the Core plugin for handling pass

4.2 Access Token Class

This class represents an access token than has been issued to a client on behalf of a user. The authenticati
2.0.

```
package com.yourapp

class AccessToken {

String authenticationKey
    byte[] authentication

String username
    String clientId

String value
    String tokenType

Date expiration

static hasOne = [refreshToken: String]
    static hasMany = [scope: String]

static constraints = {
        username nullable: true
        clientId nullable: false, blank: false
        value nullable: false, blank: false, unique: true
        tokenType nullable: false, blank: false
        expiration nullable: false
        scope nullable: false
        refreshToken nullable: true
        authenticationKey nullable: false, blank: false, unique: true
        authentication nullable: false, minSize: 1, maxSize: 1024 * 4
    }

static mapping = {
        version false
        scope lazy: false
    }
}
```

4.3 Refresh Token Class

This class represents a refresh token issued as part of one of the grants that supports issuing a refresh tok configured. See token services configuration for more. The authentication object serialized is an instance o

```
package com.yourapp

class RefreshToken {

String value
    Date expiration
    byte[] authentication

static constraints = {
        value nullable: false, blank: false, unique: true
        expiration nullable: false
        authentication nullable: false, minSize: 1, maxSize: 1024 * 4
    }

static mapping = {
        version false
    }
}
```

4.4 Authorization Code Class

This class represents an authorization code that has been issued via the authorization endpoint as p `OAuth2Authentication` from Spring Security OAuth 2.0.

```
package com.yourapp

class AuthorizationCode {

byte[] authentication
    String code

static constraints = {
        code nullable: false, blank: false, unique: true
        authentication nullable: false, minSize: 1, maxSize: 1024 * 4
    }

static mapping = {
        version false
    }
}
```

5 Optional Domain Classes

The plugin provides support for using a GORM backed `ApprovalStore` with the `ApprovalStoreU` required if the consuming application is configured to use the `UserApprovalSupport.APPROVAL_S`

The s2-init-oauth2-approval script will create the required domain class for you in a specified packag customize the generated class to fit your needs. If you change the default property names, you will need t the section on domain class properties for more information.

The below discussion assumes the s2-init-oauth2-approval script has been run with `com.yourapp` specif

5.1 Approval Class

This class represents a prior scoped approval granted to a client by a user.

```
package com.yourapp

class Approval {

String username
    String clientId

String scope
    boolean approved

Date expiration
    Date lastModified

static constraints = {
        username nullable: false, blank: false
        clientId nullable: false, blank: false
        scope nullable: false, blank: false
        expiration nullable: false
        lastModified nullable: false
    }
}
```

6 Domain Class Properties

No default class name is assumed for the required domain classes. They must be specified in `grails-ap`
The following properties exist in the `grails.plugin.springsecurity.oauthProvider` names

6.1 Client Class Properties

Property	Default Value	M
clientLookup.className	`null`	C
clientLookup.clientIdPropertyName	`'clientId'`	C
clientLookup.clientSecretPropertyName	`'clientSecret'`	C
clientLookup.accessTokenValiditySecondsPropertyName	`'accessTokenValiditySeconds'`	C
clientLookup.refreshTokenValiditySecondsPropertyName	`'refreshTokenValiditySeconds'`	C
clientLookup.authoritiesPropertyName	`'authorities'`	C
clientLookup.authorizedGrantTypesPropertyName	`'authorizedGrantTypes'`	C
clientLookup.resourceIdsPropertyName	`'resourceIds'`	C
clientLookup.scopesPropertyName	`'scopes'`	C
clientLookup.autoApproveScopesPropertyName	`'autoApproveScopes'`	C al
clientLookup.redirectUrisPropertyName	`'redirectUris'`	C
clientLookup.additionalInformationPropertyName	`'additionalInformation'`	C

6.2 Access Token Class Properties

Property	Default Value	Meaning
accessTokenLookup.className	`null`	Access token class n
accessTokenLookup.authenticationKeyPropertyName	`'authenticationKey'`	Access token class se
accessTokenLookup.authenticationPropertyName	`'authentication'`	Access token class se
accessTokenLookup.usernamePropertyName	`'username'`	Access token class us
accessTokenLookup.clientIdPropertyName	`'clientId'`	Access token class cl
accessTokenLookup.valuePropertyName	`'value'`	Access token class va
accessTokenLookup.tokenTypePropertyName	`'tokenType'`	Access token class to
accessTokenLookup.expirationPropertyName	`'expiration'`	Access token class ex
accessTokenLookup.refreshTokenPropertyName	`'refreshToken'`	Access token class re
accessTokenLookup.scopePropertyName	`'scope'`	Access token class sc

Currently only `'bearer'` tokens are supported.

6.3 Refresh Token Class Properties

Property	Default Value	Meaning
refreshTokenLookup.className	`null`	Refresh token class name.
refreshTokenLookup.authenticationPropertyName	`'authentication'`	Refresh token class serialized
refreshTokenLookup.valuePropertyName	`'value'`	Refresh token class value fie
refreshTokenLookup.expirationPropertyName	`'expiration'`	Refresh

6.4 Authorization Code Class Properties

Property	Default Value	Meaning
authorizationCodeLookup.className	`null`	Authorization code clas
authorizationCodeLookup.authenticationPropertyName	`'authentication'`	Authorization code clas
authorizationCodeLookup.codePropertyName	`'code'`	Authorization code clas

7 Configuration

The plugin is pessimistic by default, locking down as much as possible to guard against accic `grails-app/conf/Config.groovy`. The properties below exist in the `grails.plugin.sprin`

7.1 Plugin

The following properties define whether the plugin is active and where the required filters are registered in

Property	Default Value
active	`true`
filterStartPosition	`SecurityFilterPosition.X509_FILTER.order`
clientFilterStartPosition	`SecurityFilterPosition.DIGEST_AUTH_FILTER.or`
statelessFilterStartPosition	`SecurityFilterPosition.SECURITY_CONTEXT_FILT`
exceptionTranslationFilterStartPosition	`SecurityFilterPosition.EXCEPTION_TRANSLATION`
registerStatelessFilter	`true`
registerExceptionTranslationFilter	`true`
realmName	`Grails OAuth2 Realm`

7.2 Endpoint URLs

The endpoint URLs used by the underlying Spring Security OAuth 2.0 implementation can be changed usi

Property	Default Value	Meaning
authorizationEndpointUrl	`'/oauth/authorize'`	Authorization endpoint URL.
tokenEndpointUrl	`'/oauth/token'`	Token endpoint URL.
userApprovalEndpointUrl	`'/oauth/confirm_access'`	URL of the view to display for confirming
userApprovalParameter	`'user_oauth_approval'`	The name of the parameter submitted in t confirmed (`true`) or denied (`false`) acce
errorEndpointUrl	`'/oauth/error'`	URL of the view to display if an error occ the query or fragment of the client's redir URI.

When changing the URL for the `authorizationEndpointUrl` or `tokenEndpointUrl`, you **mus** default configuration as an example, your `grails-app/conf/Config.groovy` will look like this:

```
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
        '/oauth/authorize.dispatch':      ["isFullyAuthenticated() and (request.g
        '/oauth/token.dispatch':          ["isFullyAuthenticated() and request.ge
        ...
```

To change the `authorizationEndpointUrl` to `/authorize`, you will need to make the following

```
grails.plugin.springsecurity.oauthProvider.authorizationEndpointUrl = '/authorize

grails.plugin.springsecurity.controllerAnnotations.staticRules = [
        '/authorize.dispatch':            ["isFullyAuthenticated() and (request.g
        '/oauth/token.dispatch':          ["isFullyAuthenticated() and request.ge
        ...
```

The URL mapping must include the `.dispatch` suffix in order integrate with the underlying Spring MV

7.3 Token Services

The following properties apply to how tokens are issued and how long they are valid. If a client has define for the token services.

Property	Default Value	Meaning
tokenServices.accessTokenValiditySeconds	`60 * 60 * 12`	The length of time that an access
tokenServices.refreshTokenValiditySeconds	`60 * 60 * 24 * 30`	The length of time that a refresh t
tokenServices.reuseRefreshToken	`false`	Whether a new refresh token shou
tokenServices.supportRefreshToken	`true`	Whether a refresh token can be is

7.4 Supported Grant Types

The following properties determine which of the standard grant types the application can support. Individu

Property	Default Value	Meaning
grantTypes.authorizationCode	`true`	Whether the Authorization Code Grant is supported.
grantTypes.implicit	`true`	Whether the Implicit Grant is supported.
grantTypes.clientCredentials	`true`	Whether the Client Credentials Grant is supported.
grantTypes.password	`true`	Whether the Resource Owner Password Credentials is sup
grantTypes.refreshToken	`true`	Whether Refresh Token Grant is supported.

7.5 Additional Authorization Constraints

The plugin enforces the following restrictions on authorization request params:

Property	Default Value	Meaning
authorization.requireRegisteredRedirectUri	`true`	Whether a client is required to have registered a re *Authorization Code Redirection URI Manipulation*
authorization.requireScope	`true`	Whether the scope for each access token requested i

7.6 User Approval Configuration

The plugin provides support for the three `UserApprovalHandler` implementations provided by the u to configure the method of auto-approval used by the application. The following properties determine whic

Property	Default Value	Meaning
auto	`EXPLICIT`	Determines which method of auto-approval to use. The v and must be `EXPLICIT`, `TOKEN_STORE` or `APPROVAL_`
handleRevocationAsExpiry	`false`	When configured to use an approval store for auto-approv `true`) or delete the approval (`false`) outright.
approvalValiditySeconds	`60 * 60 * 24 * 30`	When configured to use an approval store for auto-approv
scopePrefix	`'scope.'`	When configured to use an approval store for auto-approv

The `auto` property determines which of the three `UserApprovalHandler` provided by Spring OAuth

The default option is to require explicit approval for every authorization and is equivalent to setting `auto`

```
grails.plugin.springsecurity.oauthprovider.approval.auto = UserApproval.EXPLICIT
```

Auto-approval based on previously issued access tokens is supported via the `TokenStoreUserAp` `TOKEN_STORE`:

```
grails.plugin.springsecurity.oauthprovider.approval.auto = UserApproval.TOKEN_STO
```

Auto-approval based on prior approvals is supported via the `ApprovalStoreUserApprovalHandle`

```
grails.plugin.springsecurity.oauthprovider.approval.auto = UserApproval.APPROVAL_
```

The plugin will configure the `TokenStoreUserApprovalHandler` and `ApprovalStoreUse` respectively.

Please consult Spring OAuth directly for more information on the usage of the `TokenStore` and `Appro`

7.7 Default Client Configuration

An application can use the following properties to define the default values that will be used when creati will not allow a client to retrieve an access token unless they have explicitly registered support for the requ

Property	Default Value	Meaning
defaultClientConfig.resourceIds	`[]`	Resources the client is authorized to ac rules.
defaultClientConfig.authorizedGrantTypes	`[]`	Grant types the client supports.
defaultClientConfig.scope	`[]`	Scope to use for each access token reque
defaultClientConfig.autoApproveScopes	`[]`	Scopes to auto-approve for authorizatio the default configuration.
defaultClientConfig.registeredRedirectUri	`null`	URI to redirect the user-agent to during
defaultClientConfig.authorities	`[]`	Roles and authorities granted to the clien
defaultClientConfig.accessTokenValiditySeconds	`null`	The length of time that an access token services if available.
defaultClientConfig.refreshTokenValiditySeconds	`null`	The length of time that a refresh token services if available.
defaultClientConfig.additionalInformation	`[:]`	Additional information about the client.

7.8 Filter Chain Configuration

Spring Security Core plugin's `securityContextPersistenceFilter` stores state in the HTTP ses

By default, the OAuth2 plugin will register the `statelessSecurityContextPersistenceFilt` Spring Security Core plugin. This is provided as a convenience for the plugin consumer, so they can remo below. This automatic filter registration can be disabled by setting the `registerStatelessFilter` c

The plugin registers an `OAuth2AuthenticationProcessingFilter` under the bean name `oau` store for resource access.

The plugin registers a `ClientCredentialsTokenEndpointFilter` under the bean name `clie` client specified in any OAuth 2.0 requests.

Finally, the plugin registers an `ExceptionTranslationFilter` under the bean name `oauth2Exc` rather than the `HttpSessionRequestCache` instance that the Spring Security Core plu `statelessSecurityContextPersistenceFilter`, this filter is registered automatically by th configuration option to `false`.

The following filter chain configuration is recommended:

```
grails.plugin.springsecurity.filterChain.chainMap = [
            '/oauth/token':
'JOINED_FILTERS,-oauth2ProviderFilter,-securityContextPersistenceFilter,-logoutFi
            '/securedOAuth2Resources/**':
'JOINED_FILTERS,-securityContextPersistenceFilter,-logoutFilter,-rememberMeAuthen
            '/**':
'JOINED_FILTERS,-statelessSecurityContextPersistenceFilter,-oauth2ProviderFilter,
   ]
```

The `oauth2ProviderFilter` and stateful `securityContextPersistenceFilter` and `exce` `securityContextPersistenceFilter` removed, the `statelessSecurityContextPersis` the `exceptionTranslationFilter` will allow the `oauth2ExceptionTranslationFilter`

The `securityContextPersistenceFilter` and `exceptionTranslationFilter` a `oauth2ProviderFilter` must not be removed, as this filter is responsible for authenticating the OAu

It is recommend that filter chain(s) for non-OAuth 2.0 resources have all OAuth 2.0 specific filt `oauth2ProviderFilter`, `clientCredentialsTokenEndpointFilter`, and `oauth2Excep` `rememberMeAuthenticationFilter` are removed from the filter chains for the token endpoint and