

Spring Security OAuth2 Plugin - Reference Documentation

Authors:

Version: 2.0-RC3

Table of Contents

- 1** Introduction to the Spring Security OAuth2 Plugin
 - 1.1** Change Log
- 2** Getting Started
 - 2.1** Install Plugin
 - 2.2** Create Domain Classes
 - 2.3** Secure Authorization and Token Endpoints
 - 2.4** Add Client Provider
 - 2.5** Exclude client_secret From Logs
 - 2.6** (Optional) Customize Error and Confirm Access Views
 - 2.7** Client Registration
 - 2.8** Controlling Access to Resources
 - 2.9** Trouble Shooting
- 3** Example Flows
 - 3.1** Authorization Code Grant
 - 3.2** Implicit Grant
 - 3.3** Resource Owner Password Credentials Grant
 - 3.4** Client Credentials Grant
 - 3.5** Refresh Token Grant
- 4** Required Domain Classes
 - 4.1** Client Class
 - 4.2** Access Token Class
 - 4.3** Refresh Token Class
 - 4.4** Authorization Code Class
- 5** Optional Domain Classes
 - 5.1** Approval Class
- 6** Domain Class Properties
 - 6.1** Client Class Properties
 - 6.2** Access Token Class Properties
 - 6.3** Refresh Token Class Properties
 - 6.4** Authorization Code Class Properties

7 Configuration

7.1 Plugin

7.2 Endpoint URLs

7.3 Token Services

7.4 Supported Grant Types

7.5 Additional Authorization Constraints

7.6 User Approval Configuration

7.7 Default Client Configuration

7.8 Filter Chain Configuration

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 on

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. The 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 classes are present.

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 Security. Add the following to `grails-app/conf/Config.groovy`:

```
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
    '/oauth/authorize.dispatch': ["isFullyAuthenticated() and (request.getRemoteHost() == 'localhost')"],
    '/oauth/token.dispatch': ["isFullyAuthenticated() and request.getRemoteHost() == 'localhost'"],
    ...
]
```

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

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

2.4 Add Client Provider

Next you will need to add the `clientCredentialsAuthenticationProvider` to the list of providers.

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

The order is important. The `clientCredentialsAuthenticationProvider` **must** occur first in the list to ensure that the client can obtain a 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 logged.

```
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', 'implicit'],
        authorities: ['ROLE_CLIENT'],
        scopes: ['read', 'write'],
        redirectUri: ['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. Additional Spring library. Refer to the methods in [OAuth2SecurityExpressionMethods](#) for what is available in the plugin.

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

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() and"])
  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 `access_token`, you must include `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 to

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

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

```
http://myredirect.com/#access_token=4e22ad4f-08ae-49dc-befb-2c9821af04dl&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 authorization.

```
http://localhost:8080/oauth2-test/oauth/token?grant_type=refresh_token&refresh_token=269afd46-0b41-45c2-a920-7d5af8a38d56
```

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

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 update the generated classes to fit your needs. If you change the default property names, you will need to update c on [domain class properties](#) for more information.



The `maxSize` constraints in the generated domain classes have been set to reasonable default values for 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.package` as the package name and `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 for

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,
        redirectUri: 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

        redirectUri 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 token configured. See [token services configuration](#) for more. The authentication object serialized is an instance of

```

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 part of 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 `ApprovalStoreUserApprovalSupport` required if the consuming application is configured to use the `UserApprovalSupport.APPROVAL_STORE`.

The [s2-init-oauth2-approval](#) script will create the required domain class for you in a specified package; customize the generated class to fit your needs. If you change the default property names, you will need to update 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` specified.

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-app`. The following properties exist in the `grails.plugin.springsecurity.oauthProvider` namespace.

6.1 Client Class Properties

Property	Default Value	Meaning
<code>clientLookup.className</code>	<code>null</code>	Client class name
<code>clientLookup.clientIdPropertyName</code>	<code>'clientId'</code>	Client class id property name
<code>clientLookup.clientSecretPropertyName</code>	<code>'clientSecret'</code>	Client class secret property name
<code>clientLookup.accessTokenValiditySecondsPropertyName</code>	<code>'accessTokenValiditySeconds'</code>	Client class access token validity seconds property name
<code>clientLookup.refreshTokenValiditySecondsPropertyName</code>	<code>'refreshTokenValiditySeconds'</code>	Client class refresh token validity seconds property name
<code>clientLookup.authoritiesPropertyName</code>	<code>'authorities'</code>	Client class authorities property name
<code>clientLookup.authorizedGrantTypesPropertyName</code>	<code>'authorizedGrantTypes'</code>	Client class authorized grant types property name
<code>clientLookup.resourceIdsPropertyName</code>	<code>'resourceIds'</code>	Client class resource ids property name
<code>clientLookup.scopesPropertyName</code>	<code>'scopes'</code>	Client class scopes property name
<code>clientLookup.autoApproveScopesPropertyName</code>	<code>'autoApproveScopes'</code>	Client class auto approve scopes property name
<code>clientLookup.redirectUriPropertyName</code>	<code>'redirectUri'</code>	Client class redirect uri property name
<code>clientLookup.additionalInformationPropertyName</code>	<code>'additionalInformation'</code>	Client class additional information property name

6.2 Access Token Class Properties

Property	Default Value	Meaning
<code>accessTokenLookup.className</code>	<code>null</code>	Access token class name
<code>accessTokenLookup.authenticationKeyPropertyName</code>	<code>'authenticationKey'</code>	Access token class authentication key property name
<code>accessTokenLookup.authenticationPropertyName</code>	<code>'authentication'</code>	Access token class authentication property name
<code>accessTokenLookup.usernamePropertyName</code>	<code>'username'</code>	Access token class username property name
<code>accessTokenLookup.clientIdPropertyName</code>	<code>'clientId'</code>	Access token class client id property name
<code>accessTokenLookup.valuePropertyName</code>	<code>'value'</code>	Access token class value property name
<code>accessTokenLookup.tokenTypePropertyName</code>	<code>'tokenType'</code>	Access token class token type property name
<code>accessTokenLookup.expirationPropertyName</code>	<code>'expiration'</code>	Access token class expiration property name
<code>accessTokenLookup.refreshTokenPropertyName</code>	<code>'refreshToken'</code>	Access token class refresh token property name
<code>accessTokenLookup.scopePropertyName</code>	<code>'scope'</code>	Access token class scope property name

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 accidental changes to `grails-app/conf/Config.groovy`. The properties below exist in the `grails.plugin.springsecurity` namespace.

7.1 Plugin

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

Property	Default Value
<code>active</code>	<code>true</code>
<code>filterStartPosition</code>	<code>SecurityFilterPosition.X509_FILTER.order</code>
<code>clientFilterStartPosition</code>	<code>SecurityFilterPosition.DIGEST_AUTH_FILTER.order</code>
<code>statelessFilterStartPosition</code>	<code>SecurityFilterPosition.SECURITY_CONTEXT_FILTER.order</code>
<code>exceptionTranslationFilterStartPosition</code>	<code>SecurityFilterPosition.EXCEPTION_TRANSLATION_FILTER.order</code>
<code>registerStatelessFilter</code>	<code>true</code>
<code>registerExceptionTranslationFilter</code>	<code>true</code>
<code>realmName</code>	<code>Grails OAuth2 Realm</code>

7.2 Endpoint URLs

The endpoint URLs used by the underlying Spring Security OAuth 2.0 implementation can be changed using the following properties.

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 the confirmed (true) or denied (false) access
errorEndpointUrl	' /oauth/error '	URL of the view to display if an error occurs in the query or fragment of the client's redirect URI.

When changing the URL for the `authorizationEndpointUrl` or `tokenEndpointUrl`, you **must** use the 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.getHeader('X-Requested-With') == 'XMLHttpRequest')"],
    '/oauth/token.dispatch':    ["isFullyAuthenticated() and request.getHeader('X-Requested-With') == 'XMLHttpRequest'"],
    ...
]
```

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

```
grails.plugin.springsecurity.oauthProvider.authorizationEndpointUrl = '/authorize'
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
    '/authorize.dispatch': ["isFullyAuthenticated() and (request.getHeader('X-Requested-With') == 'XMLHttpRequest')"],
    '/oauth/token.dispatch': ["isFullyAuthenticated() and request.getHeader('X-Requested-With') == 'XMLHttpRequest'"],
    ...
]
```

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

7.3 Token Services

The following properties apply to how tokens are issued and how long they are valid. If a client has defined token services, these properties will be used.

Property	Default Value	Meaning
tokenServices.accessTokenValiditySeconds	60 * 60 * 12	The length of time that an access token is valid.
tokenServices.refreshTokenValiditySeconds	60 * 60 * 24 * 30	The length of time that a refresh token is valid.
tokenServices.reuseRefreshToken	false	Whether a new refresh token should be issued when the current one expires.
tokenServices.supportRefreshToken	true	Whether a refresh token can be issued.

7.4 Supported Grant Types

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

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 supported.
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 redirect URI. <i>Authorization Code Redirection URI Manipulation</i>
authorization.requireScope	true	Whether the scope for each access token requested is required.

7.6 User Approval Configuration

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

Property	Default Value	Meaning
auto	EXPLICIT	Determines which method of auto-approval to use. The value must be EXPLICIT, TOKEN_STORE or APPROVAL_STORE.
handleRevocationAsExpiry	false	When configured to use an approval store for auto-approval (true) or delete the approval (false) outright.
approvalValiditySeconds	60 * 60 * 24 * 30	When configured to use an approval store for auto-approval, the validity of the approval in seconds.
scopePrefix	'scope.'	When configured to use an approval store for auto-approval, the prefix for the scope.

The `auto` property determines which of the three `UserApprovalHandler` provided by Spring OAuth2

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 `TokenStoreUserApprovalHandler`. To configure this, set `auto` to `TOKEN_STORE`:

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

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 `ApprovalStoreUser` 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 creating a client. The plugin will not allow a client to retrieve an access token unless they have explicitly registered support for the requested scope.

Property	Default Value	Meaning
<code>defaultClientConfig.resourceIds</code>	<code>[]</code>	Resources the client is authorized to access. If empty, the client is authorized to access all resources.
<code>defaultClientConfig.authorizedGrantTypes</code>	<code>[]</code>	Grant types the client supports.
<code>defaultClientConfig.scope</code>	<code>[]</code>	Scope to use for each access token request.
<code>defaultClientConfig.autoApproveScopes</code>	<code>[]</code>	Scopes to auto-approve for authorization requests using the default configuration.
<code>defaultClientConfig.registeredRedirectUri</code>	<code>null</code>	URI to redirect the user-agent to during the authorization process.
<code>defaultClientConfig.authorities</code>	<code>[]</code>	Roles and authorities granted to the client.
<code>defaultClientConfig.accessTokenValiditySeconds</code>	<code>null</code>	The length of time that an access token is valid for. If null, the default value is 3600 seconds.
<code>defaultClientConfig.refreshTokenValiditySeconds</code>	<code>null</code>	The length of time that a refresh token is valid for. If null, the default value is 36000 seconds.
<code>defaultClientConfig.additionalInformation</code>	<code>[:]</code>	Additional information about the client.

7.8 Filter Chain Configuration

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

By default, the OAuth2 plugin will register the `statelessSecurityContextPersistenceFilter` with the Spring Security Core plugin. This is provided as a convenience for the plugin consumer, so they can remove the filter from the filter chain if they do not need it. This automatic filter registration can be disabled by setting the `registerStatelessFilter` property to `false`.

The plugin registers an `OAuth2AuthenticationProcessingFilter` under the bean name `oauth2AuthenticationProcessingFilter` to handle the authentication process for resource access.

The plugin registers a `ClientCredentialsTokenEndpointFilter` under the bean name `clientCredentialsTokenEndpointFilter` to handle the client credentials grant type. The filter will only be active if the client is 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 plugin registers. If the `statelessSecurityContextPersistenceFilter` is registered automatically by the configuration option to `false`.

The following filter chain configuration is recommended:

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

The `oauth2ProviderFilter` and stateful `securityContextPersistenceFilter` and `exceptionTranslationFilter` are removed, the `statelessSecurityContextPersistenceFilter` is added, and the `exceptionTranslationFilter` will allow the `oauth2ExceptionTranslationFilter` to handle exceptions.

The `securityContextPersistenceFilter` and `exceptionTranslationFilter` are added, and the `oauth2ProviderFilter` must not be removed, as this filter is responsible for authenticating the OAuth 2.0 resources.

It is recommended that filter chain(s) for non-OAuth 2.0 resources have all OAuth 2.0 specific filters removed. The `oauth2ProviderFilter`, `clientCredentialsTokenEndpointFilter`, and `oauth2ExceptionTranslationFilter` are removed from the filter chains for the token endpoint and