# OGC Testbed-13

***Security ER***

# Table of Contents

Publication Date: 2018-01-11

Approval Date: 2017-12-07

Posted Date: 2017-11-13

Reference number of this document: OGC 17-021

Reference URL for this document: http://www.opengis.net/doc/PER/t13-AB002

Category: Public Engineering Report

Editor: Andreas Matheus

Title: OGC Testbed-13: Security ER

---

**OGC Engineering Report**

**COPYRIGHT**



**WARNING**

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

**LICENSE AGREEMENT**

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

# Chapter 1. Summary

The Security Engineering Report (ER) covers two Testbed 13 topics:

- The implementation of authentication and authorization plugins for the QGIS open source desktop GIS client and
- the implementation of secured workflow.

The authentication plugins implement the SAML2 ECP with PAOS binding and IdP discovery from the SAML2 federation metadata URL. The access right delegation plugin implements applicable OAuth2 grant types.

Regarding the first topic, this ER discusses the "fit for purpose" aspects for the OAuth2 and SAML2 in the context of an open source desktop application. It also covers the QGIS development as well as building and deployment aspects. Most of the work related to this topic was provided by Secure Dimensions.

Regarding the second topic, this ER outlines the architecture approach and the implications to implementations for security in OGC service workflows as well as the implementation approach itself. Most of the work related to this topic was provided by 52°North.

## 1.1. Requirements

The requirements addressed by this ER are the implementation of access right delegation (OAuth2) plugin for QGIS as an extension to version 2.18.4. These are able to use OAuth2 protected services WMS and WMTS provided by Dutch Cadaster.

The requirement for workflow security is also to document the following use cases, as implemented by 52°North:

- Use Case 1: Dominating Privileges
- Use Case 2: Tunneling Proxies
- Use Case 3: Identity Mediation

## 1.2. Prior-After Comparison

The security state before the Testbed 13 was that the QGIS Open Source desktop GIS was capable to connect to OAuth2 protected OGC Web Services, but a manual registration with Authorization Servers was required.

QGIS was not able to connect to OGC Web Services (OWS) protected with version 2 of the Security Assertion Markup Language (SAML). It was also not possible to construct workflows of protected OGC Web Services.

After Testbed 13, the Open Source desktop GIS QGIS can be used to connect to protected services leveraging OAuth2 and the plugin registration is simplified by the use of digitally signed software statements and Dynamic Client Registration. Now, it is also possible to connect to SAML2 protected

OGC Web Services that support the SAML2 Enhanced Client Proxy Protocol (ECP). In workflows, different security recommendations (in addition to those exposed in Testbed 12) will enable the construction and execution of workflows comprised of secured OGC services.

## 1.3. What does this ER mean for the Working Group and OGC in general

This ER has security applicability (implications) to different OGC working groups, as security is a crosscutting theme across the entire OGC domain.

For the OGC Security Domain Working Group (DWG), this ER provides results towards the use of "modern" security solutions like OAuth2 and SAML2 which expands the generally simple authentication, such as HTTP BASIC authentication. It should thereby stimulate the use of modern security in infrastructures to make protected data sets available via OGC Web Services. Beside the technical merit, this should provide new security based business opportunities.

For the OWS Common Security Standards Working Group (SWG), the implementation of the security extension to QGIS will enable testing the described approach of annotated capabilities regarding aspects like 'fit for purpose'. The feedback will be valuable to the SWG in the process of standardizing the interoperability aspects.

The aspects of workflow security may introduce novel approaches to develop and execute workflows of secured services. It should stimulate discussions in the Workflow DWG and the Security DWG regarding the 'fit for purpose' and interoperability aspects.

## 1.4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

*Table 1. Contacts*

| Name | Organization |
|---|---|
| Andreas Matheus | University of the Bundeswehr |
| Benjamin Pross | 52°North GmbH |
| Christoph Stasch | 52°North GmbH |

## 1.5. Future Work

It is expected that this ER stimulates future work regarding the OGC to standardize requirements for client applications to work on modern security aspects and the work being standardized by the OWS Common SWG. More details can be found in section "Conclusions and Future Work".

## 1.6. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any

or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

# Chapter 2. References

The following normative documents are referenced in this document.

*NOTE: Only normative standards are referenced here, e.g. OGC, ISO or other SDO standards. All other references are listed in the bibliography.*

- OGC 06-121r9, OGC® Web Services Common Standard, April 2010 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC 14-065, OGC® WPS 2.0 Interface Standard, March 2015 [http://docs.opengeospatial.org/is/14-065/14-065.html]
- IETF, RFC 6749 - The OAuth 2.0 Authorization Framework [https://tools.ietf.org/html/rfc6749]
- IETF, RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage [https://tools.ietf.org/html/rfc6750]
- IETF, RFC 6819 - OAuth 2.0 Threat Model and Security Considerations [https://tools.ietf.org/html/rfc6819]
- IETF, RFC 7009 - OAuth 2.0 Token Revocation [https://tools.ietf.org/html/rfc7009]
- IETF, RFC 7235 - Hypertext Transfer Protocol (HTTP/1.1): Authentication [https://tools.ietf.org/html/rfc7235]
- IETF, RFC 7515 - JSON Web Signature JWS [https://tools.ietf.org/html/rfc7515]
- IETF, RFC 7519 - JSON Web Token (JWT) [https://tools.ietf.org/html/rfc7519]
- IETF, RFC 7591 - OAuth 2.0 Dynamic Client Registration Protocol [https://tools.ietf.org/html/rfc7591]
- IETF, RFC 7592 - OAuth 2.0 Dynamic Client Registration Management Protocol [https://tools.ietf.org/html/rfc7592]
- IETF, RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients [https://tools.ietf.org/html/rfc7636]
- IETF, RFC 7662 - OAuth 2.0 Token Introspection [https://tools.ietf.org/html/rfc7662]
- OASIS - SAML2 Profiles [https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf]
- OASIS - SAML2 Bindings [https://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf]

# Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

## 3.1. Authorization Server

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.[RFC 6749]

## 3.2. Resource Server

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.[RFC 6749]

## 3.3. Access Token

Access tokens are credentials used to access protected resources.[RFC 6749]

## 3.4. QGIS

A Free and Open Source Geographic Information System[http://qgis.org]

# Chapter 4. Abbreviated terms

*NOTE: The abbreviated terms clause gives a list of the abbreviated terms and the symbols necessary for understanding this document. All symbols should be listed in alphabetical order.Some more frequently used abbreviated terms are provided below as examples.*

- API Application Programming Interface
- AS Authorization Server
- ECP Enhanced Client Proxy Profile
- RS Resource Server
- OAuth OAuth
- OIDC OpenID Connect
- RFC Request For Comments
- OP OpenID Connect Identity Provider
- OWS OGC Web Service
- SAML Security Assertion Markup Language
- SSO Single-Sign-On

# Chapter 5. Overview

This public engineering report is being produced during the OGC Testbed 13, executed from April to December 2017. Even though the title is broad, this document focuses on specific security topics that were worked on during Testbed 13.

This document contains the following information:

- Introduction / Overview to the OAuth2 / OpenID Connect and SAML2 standards as they are key to the activities covered by the ER.
- Detailed discussion on the use of OAuth2 with Open Source desktop applications, such as QGIS
- The use of the QGIS plugin for OAuth2 and the options to register the plugin with OAuth2 Authorization Servers
- The use of the QGIS plugin for SAML2
- How to compile and install the QGIS plugins for OAuth2 and SAML2
- Introduction and details on implementation of security in workflows based on the OGC Web Processing Service (WPS)
- Technology Integration tests regarding the QGIS plugin for OAuth2

This document was reviewed and approved by the OGC Security DWG on November xx, 2017

| | |
|---|---|
| **NOTE** | *Note*<br>Please note that URL provided in this document will not last forever. It is the responsibility of the participant to keep the URL active until 6 months after the end of Testbed 13; which is 1st July 2018. |

# Chapter 6. Overview to Standards leveraged in this ER

| | |
|---|---|
| **NOTE** | *Note*<br>This section provides a brief overview to security standards used in the testbed. For any detail not presented, the interested reader should become familiar with the standard itself by downloading a copy from the standardization organization. The URLs can be found in the References section. |

## 6.1. SAML (Security Assertion Markup Language)

The Security Assertion Markup Language (SAML) is a standard of the Organization for the Advancement of Structured Information Standards (OASIS), an alliance partner of the OGC. SAML is concerned with authentication and assertion exchange in trusted distributed systems to support Single-Sign-On and Federated Identity Management.

Many years ago, the OGC Shibboleth Interoperability Experiment conducted a study and prototype implementation of SAML with OGC Web Services. For more information on SAML and how to apply it to OGC Web services, please follow the link to the Public Engineering Report provided in the Bibliography chapter.

## 6.2. OAuth2

First, one should stress what OAuth2 is **not** about: It is **not** about authentication and it is also **not** about authorization.

Instead, the OAuth2 specification describes a framework that enables users to delegate a (controlled) set of rights to an application so that it can access the users' resources. The cool thing about the delegation is that the users must not share their original login credentials with an application. Instead, the users approve the release of a - so called - access token to the application that "carries" the approved set of rights. The application can then present the access token with the actual request to access the user’s resources. The user is the actor in OAuth2 called Resource Owner.

The most dominant use cases concentrate around social platforms like Facebook, Twitter, etc. where the users grant a particular application access to user profile, emails, blogs, etc. It is not relevant who created the application, but it is important that the user trusts the application. As the user cannot establish bi-lateral trust, the OAuth2 specification introduces a 3rd party called Authorization Server.

The Authorization Server is the actor that allows application developers to register and manage their applications. Once registered, the Authorization Server is capable of releasing access tokens to the application once granted by the user. But before the user can approve the release of an access token (which keeps a certain set of rights), the user must authenticate with the Authorization Server. How this authentication takes place is outside the OAuth2 specification. But it is mandatory that the authentication provides the Authorization Server with sufficient information about the

user.

The third actor in the OAuth2 framework is the Resource Server. This actor is responsible for authorizing requests from applications to user resources. Basically, the Resource Server must first check if the access token is valid; and if it is, check whether the requested action on the resource was previously approved by the user. OAuth2 manages the associations between different rights and access tokens via scopes. If the request to the resource is involving a scope that is carried by the attached access token, then the resource is returned. In case that the application does ask for an action on the resource which is not covered by the scopes associated with the access token, then the Resource Server returns an error.

From a business perspective, OAuth2 enables a market place where developers can build and register applications that can be run by users to do "cool" things with their resources like pictures, emails, messages etc. Business takes place if a user decides to run an application and either pays money for the use or the application displays advertisements. Either way, it is attractive for developers to register applications with an Authorization Server to make business. The OAuth2 trust model is exactly built for that: Individual developers register applications that can be trusted and run by users to do cool things with their data.

This ER would not be about security if it did not mention the dark side of the things: If the user trusts a malicious application, all kinds of attacks can be thought of that span from remote control to the user’s data to forged (money) transactions in particular in scenarios with one-click-payments enabled.

### 6.2.1. Example - Email Alert Application

The Email Alert Application can be used to illustrate the OAuth2 framework in more detail and to show the limitations of OAuth2.

The Email Alert Application shall be an application that runs on a mobile device that processes user emails and generates alerts when certain conditions on emails occur. For example, the user could configure an alert that if receiving emails on the OGC TB13 from the initiative director after business hours (which quite often happens for European users) to initiate an automatic reply stating that it is off-hours. To bring the Email Alert Application to live, we need a developer for the application and an Authorization Server. Say that Secure Dimensions implemented the application and registered it with the Google Play Store. Once the user has installed and started the application, it will ask for setting up trigger conditions. Once the user has done that, the application will ask the user for permission to access the emails stored at GMail: "Please delegate access rights to read your emails stored at GMail and send emails under your name". Once the user has approved this delegation, the application can start fetching emails and check for the alert condition.

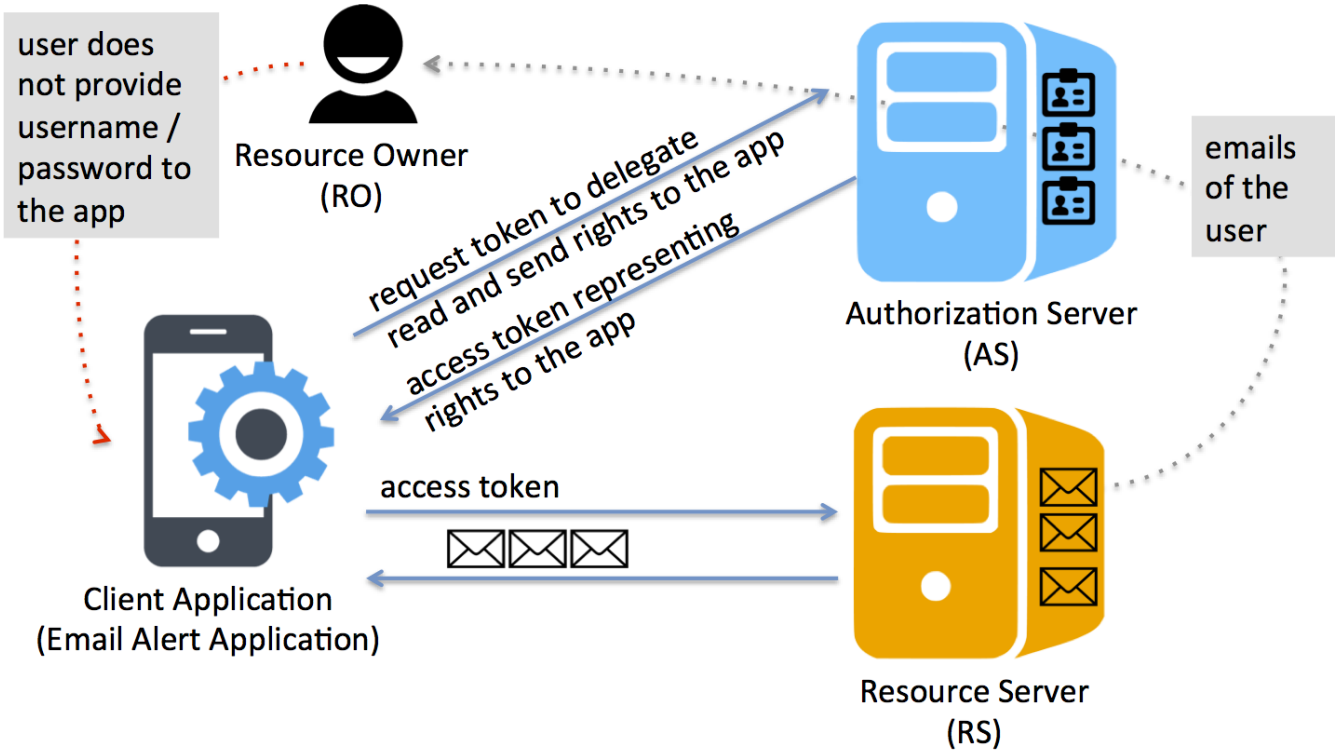


*Figure 1. Email Alert Application*

This version of the application has one important shortcoming caused by the nature of OAuth2: The application has no information about the user. The access token approved by the user **just** carries the rights for the application to access the emails of the user but there is no information about the user! Therefore, it is for example not possible for the application - unless it received the information from the user itself - to respond with proper greetings at the end such as "regards Andreas". In other words, with the information from OAuth2, the application could only send a generic message "Your email was received during off-hours. I will start processing my emails tomorrow starting 8 o'clock UTC". Please note that at this point we are stressing the lack of user information, making it difficult to motivate the OAuth2 upgrade to OpenID Connect.

## 6.3. OpenID Connect

The OpenID Connect specification is an extension to the OAuth2 framework that enables an Authorization Server to also release user information to the application. For the sake of differentiation, an OpenID Connect compliant Authorization Server is called an OpenID Connect Provider or OIDC Provider (OP). The compliance requires that three requirements are implemented: (i) The OP releases the so-called ID Token, which is a JSON Web Token that contains information about the user that delegates access rights to the application; (ii) the ID Token follows the structure defined in the specification; and (iii) the OP operates the UserInfo endpoint. Without going into details, OIDC also introduces additional flows for how the OP can release tokens to the applications. The default is that the OP releases the ID token with the access token or authentication code. This enables the application to establish a user session from the start of the flow, which can be quite handy as we will see with the example Email Alert Service.

Even though OIDC introduces authentication to the OAuth2 flow, one must keep in mind that it is just an extension and therefore is restricted to the OAuth2 framework and the specified flow. However, the introduction of the hybrid flows provides more flexibility and control to the application when requesting tokens.

An OP can basically release three different types of tokens:

- id_token: OIDC specific token that contains claims (information) about the current user
- access_token: OAuth2 token that carries the delegated rights
- refresh_token: An OAuth2 token that technically enables the application to request new (fresh) access tokens from the OP (AS respectfully) once the issued access token is expired.

### 6.3.1. Example - Email Alert Application Plus

The Email Alert Application is now split into two parts: The client part installed on the user's mobile device and a service back-end that can store conditions and process emails to raise alerts. This application is now registered with an OIDC compliant Authorization Server - hence an OP.

With the user's consent that the application can process the user's emails, the OP releases an access token (as before) but now as the duty to OIDC, it also releases an ID token. The ID token can then be used by the application to establish a user session with the backend service as well as sending emails with proper greetings (assuming that the ID token contained the user's name).

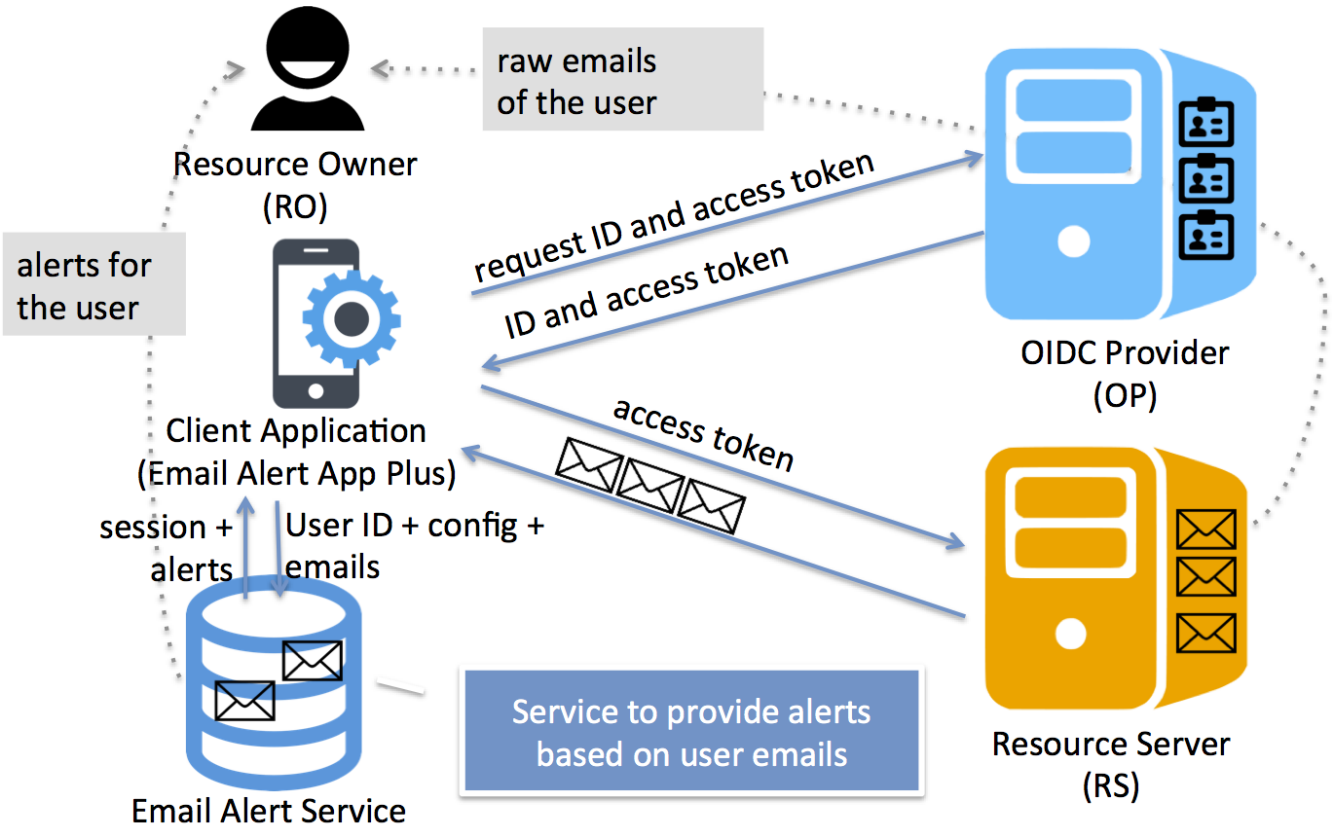


*Figure 2. Email Alert Application Plus*

### 6.3.2. OIDC UserInfo endpiont

In OAuth2, the access (and the refresh) tokens by default are opaque to the application and the Resource Server. That means that it always requires the Authorization Server to validate the access token or to obtain any more information associated with the token.

In that respect, RFC 7662 "OAuth 2.0 Token Introspection" introduces an Authorization Server endpoint that allows the Resource Server to fetch token specific metadata. Some information

provided might also be about the user. But it is important to note that this endpoint was not introduced to return user information such as profile information, emails, pictures, messages, etc. The OpenID Connect specification introduces the UserInfo endpoint for exactly that purpose.

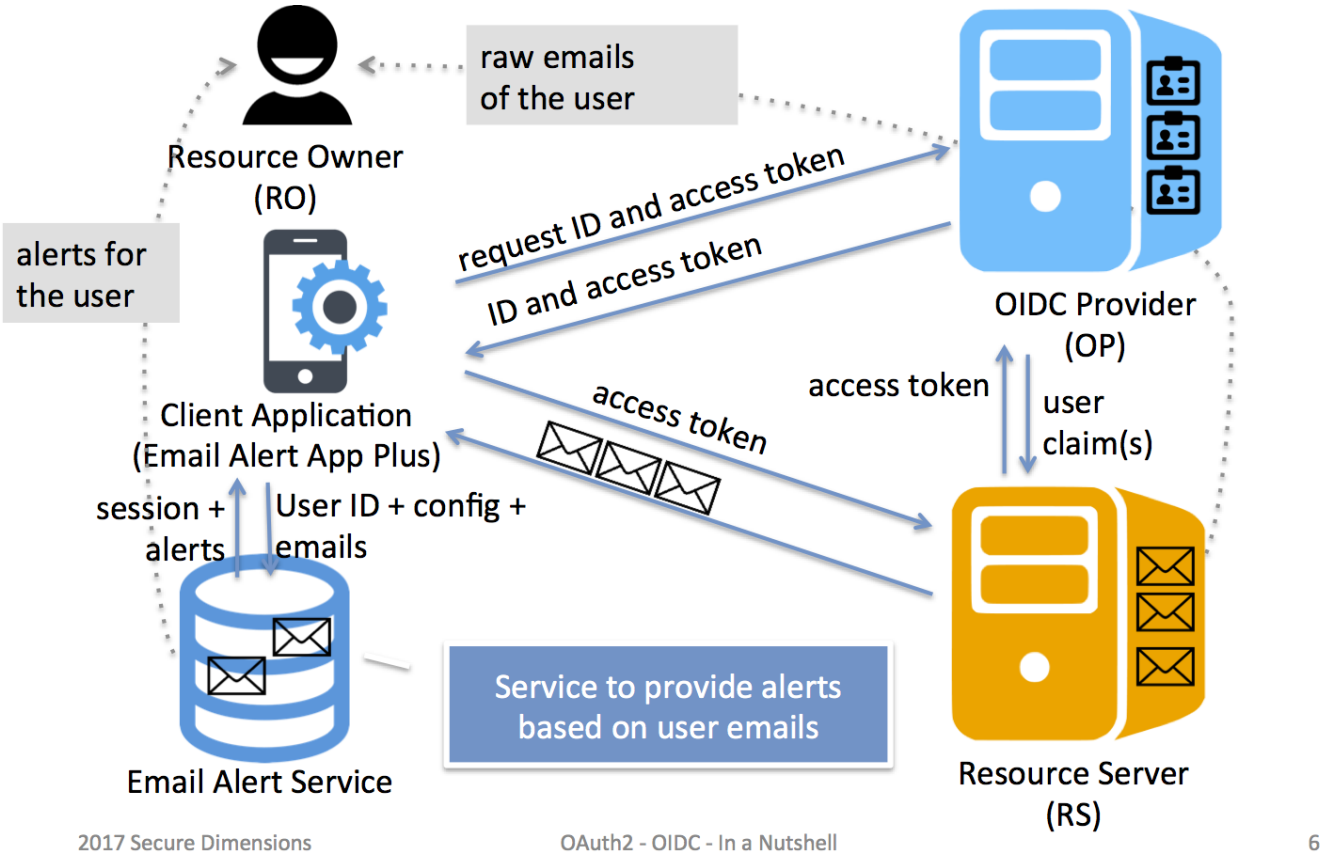


*Figure 3. Email Alert Application Plus*

Because an ID token can contain sensitive or personal information about the user, great care must be taken when issuing a token. On the other hand, the receiving party must be able to trust the information provided. In that sense, the OpenID connect specification introduces the concept of a JSON Web Token (JWT) which is digitally signed by the releasing OP. And in order to ensure that the delivery of the user information is caused to the right application, the ID token gets released with the access token (or with the authorization code leveraging the hybrid flow) in the same HTTP message.

Now, the application has user information, but the Resource Server does not as only the access token gets submitted to the Resource Server. With an OIDC compliant Authorization Server, the Resource Server can now request user information via the UserInfo endpoint. Important with this option is to note that the access token is a Bearer token per definition. That means it is like cash. If one "finds" a valid access token, the Resource Server will accept it (assuming that the request fits the delegated rights). The use of a "found" token with the UserInfo endpoint would mean disclosure of the user's information to potentially phishing parties.

The detail of user information, that a registered Resource Server can obtain from the OP depends on the scopes approved by the user; the scopes are linked with the access token. For example, the OpenID Connect specification defines different scopes that release different level of detail regarding the user information:

- profile: public profile information

- email: the user's email address
- phone: The user's phone number
- address: The user's address

Each scope is then linked with a defined set of claims that the UserInfo endpoint is going to return. Please consult the OpenID Connect Specification for more details [1: http://openid.net/developers/specs/].

The Resource Server can process the user information to fulfill a defined need. This can vary from logging of access information, to user based access control or invoicing.

# Chapter 7. OAuth2 Plugin

| | |
|---|---|
| **NOTE** | *Note*<br>This section contains the contributions from Secure Dimensions regarding the QGIS plugin for OAuth2. |

This section provides a fit for purpose discussion of OAuth2 to be used for open source desktop applications.

## 7.1. The Trust Principle for OAuth2

OAuth2 as standardized by RFC 6749 is a (access rights) delegation framework that enables a resource owner to grant (restricted) access rights to the own resources for trusted applications without providing the original access credentials, e.g. username and password to the application. As outlined in the overview section, the user's trust in the application is essential. As OAuth2 is originally designed for online applications (either running inside the user-agent - aka the web browser - or hosted on a web server), the question arrives whether the OAuth2 framework can be applied to open source desktop applications.

The essential part is how the application and the Authorization Server establish trust. As defined in RFC 6749, this is different for public and confidential clients. The question now is whether QGIS and the OAuth2 plugin qualifies for either the public client or confidential client type.

### 7.1.1. OAuth2 Trust for public clients

For a public client - aka a user-agent hosted application typically implemented in JavaScript - the application does not have a client_secret as it is unable to securely store it. For a public client, the Authorization Server establishes trust via the application's client_id and the redirect_uri. To be precise, this type of application can only receive an access token by leveraging one of the redirect URIs, registered with the Authorization Server. The supported grant types are the Authorization Code or Implicit. As standardized in RFC 6749, the implicit flow returns the access token via the redirect_uri whereas the authorization code flow returns the authorization code via the redirect_uri.

### 7.1.2. OAuth2 Trust for confidential clients

Compared to public clients, a confidential client has the ability to keep its authentication credentials safely. Therefore, this type of client application qualifies for any OAuth2 Grant Type that relies on this feature.

For a desktop application, a type of encrypted storage could be used to keep the application credentials. The keystore could keep the client_id, the client_secret and optionally the redirect_uri for a given Authorization Server. It could also store the refresh token to remember the user decision to grant access rights to the application.

As a confidential client application, the implementation of the Resource Owner Password Credentials Grant and Authorization Code Grant via a trusted online redirect URL can be

considered fit for purpose.

## 7.2. OAuth2 Trust for QGIS

But what about QGIS as an open source desktop application? What about the redirect URI for the QGIS OAuth2 plugin? Clearly, QGIS does not have an online redirect URL where the Authorization Server can redirect to in order to deliver the authorization code (Authorization Code Flow) or the access token (Implicit Flow). The only similar option would be that QGIS establishes a localhost port to listen on for the redirect. But, this would turn the client machine in a web server type of machine, as it has open port(s) listening to HTTP traffic.

The other option regarding the redirect URI would be that for the QGIS application, a redirect URI based on a custom scheme gets registered with the Operating System. So for example, QGIS could register the scheme "qgis:" with the operating system and then register the redirect_uri to start with this scheme. So for example 'qgis:callback', where the scheme qgis would trigger to start the QGIS application and call the function 'callback' that would then read the authorization code or the access token from the redirect URL.

Assuming even this approach would work on any operating system, it would reset the QGIS application state as it re-starts the application. This is certainly not ideal if the user has multiple services connected and is enjoying a particular view when trying to add an OAuth2 enabled service. Then losing the entire application state would require to "start from scratch".

An alternative approach would be using a generic online URL that can only be used with the Authorization Code Flow to display the authorization code issued by the Authorization Server. With this approach, the user must copy and paste the authorization code from the Web Browser window into the QGIS plugin. This approach would ensure that the QGIS application state can be maintained as the authentication code could be pasted into the configuration window currently used by the user to setup the service connection. This approach also does not require that the client computer must open a listening port. But on the other hand, this is a manual step for the user, requiring a copy and paste action.

It is very important to note that this approach shall not be used for the Implicit Grant as the Authorization Server would return an access token to the generic URL. Also, the Implicit Grant is not fit for purpose, as the Authorization Server does not release a refresh token with the Implicit Grant. Therefore, the plugin would need to initiate a Web Browser interaction each time the access token expires. This is - despite the fact that the access token is exposed in the clear - not user friendly.

## 7.3. Registration of the OAuth2 Plugin

Regardless which grant type the OAuth2 plugin is going to be using, the plugin must be registered with the Authorization Server in charge for the Resource Server, before it can obtain token(s). In principle, there are two different options: (i) manual registration and (ii) API based registration.

The normal process is to register the application manually with the Authorization Server. This is typically done by the developer of the application and requires an out-of-band communication with the administrator of the Authorization Server or an admin login. For Web Applications, the

developer typically has a login into the Authorization Server that allows the registration of the application. Once the application is online, any user can interact.

But this procedure is not fit for purpose for QGIS as the OAuth2 plugin as each user would have to register the plugin with all the Authorization Servers out there (and that seems to be impractical). Therefore, each user would have to register their own copy of the QGIS AOuth2 plugin with each Authorization Server that is in charge of the protected services hosted at a Resource Server. Given the number of QGIS users and installations, that seem to be an overwhelming number of client registrations for each Authorization Server.

When configuring an OGC Web Service connector to use OAuth2 based authorization, then the OAuth2 plugin should be able to register itself with the involved Authorization Server. At this point, the plugin no longer exists as Open Source but as compiled and executable code. This has an implication to the design of the OAuth2 plugin: It must be able to add this plugin as a compiled module to QGIS, like Apache Web Server modules. Using this approach, the plugin can be compiled independent from the QGIS main application. This is important as the trust with the Authorization Server can be established with the compiling entity of the plugin.

With this trust in place, the QGIS Oauth2 plugin can be registered via an API based registration mechanism, based on digitally signed client metadata - the QGIS OAuth2 plugin software statement. Each entity that compiles the OAuth2 plugin can include into the binary a software statement, that is digitally signed with the entity's private key. And the associated public key can be made available on the entity's web server. It is also possible that the entity is using a X.509 code signing certificate that can be used by any Authorization Server to validate the software statement used for automatic registration.

An alternative to the described registration process based on a software statement compiled into the plugin is to import a provided software statement. If the plugin supports the import functionality, a trusted software statement would not have to be compiled into the plugin, but the user could load the trusted software statement instead. Because the software statement is digitally signed, this process is equivalent to using a built-in software statement.

## 7.4. Dynamic Client Registration (RFC 7591)

The ability of an OAuth2 Authorization Server or OIDC Provider to support dynamic client registration via an API is defined in RFC 7591. Basically, three options exist to use the API:

- The client leverages a bearer token, previously released by the AS/OP for the purpose of dynamic client registration (how the client can obtain the token is out of scope);
- The client leverages a digitally signed software statement (a JWT) to register the application
- The AS/OP operate the registration endpoint with no authentication or token requirement, but might enforce rate limiting or throttling to prevent DoS attacks.

For Testbed 13, it was recommended implementing the AS/OP supports dynamic client registration via an open API (rate limit enforced) based on a digitally signed software statement. The AS/OP MUST only accept registration requests based on a valid digital signature on the software statement.

The feature "Dynamic Client Registration" can be implemented by any OAuth2 Authorization Server (and therefore any OIDC Provider) by implementing the RFC 7591.

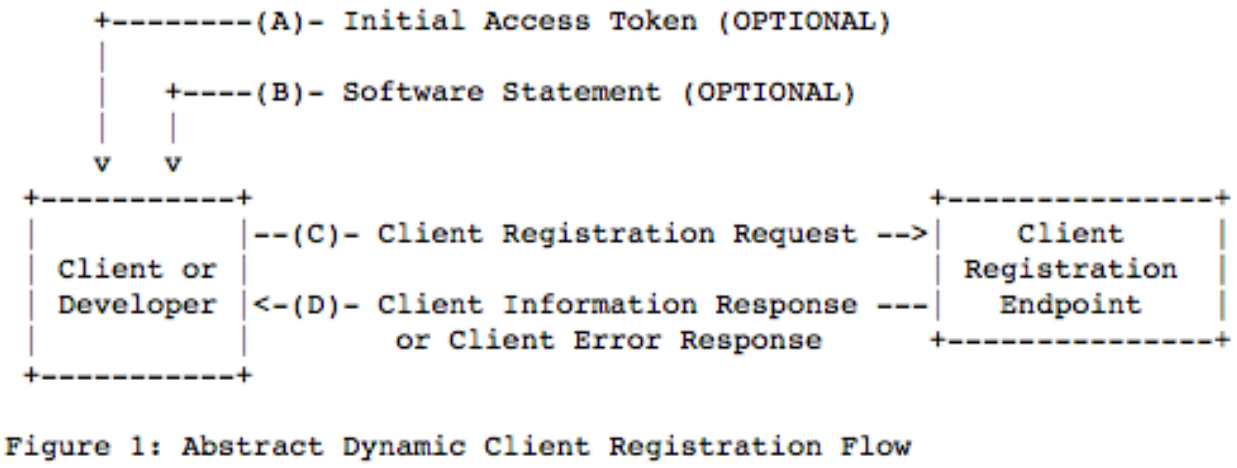


*Figure 4. Dynamic Client Registration*

The actual request for client registration is labeled "C" and includes the client metadata. Two pieces of optional information can be used by the client to commence the registration step: (i) an initial Access Token (obtained from the Authorization Server) and (ii) a Software Statement (the client metadata). If the use of an initial access token is required or whether its possible to use a software statement depends on the actual implementation of the API for dynamic client registration, provided by a particular Authorization Server.

## 7.5. Registration of the OAuth2 Plugin in TB13

As discussed earlier, open source software prevents that the developer can implement any trust in the OAuth2 plugin itself. For TB13, assume that the Authorization Server accepts a client registration based on a digitally signed software statement. The Authorization Server is able to verify the digital signature on the software statement before accepting the registration request. Following good practice, it is recommended that the public key is bundled to a code signing certificate that enables the Authorization Server to verify the digital signature on the software statement and identify the signing entity. Based on a white listing, the Authorization Server can accept a registration for any trusted entities.

For Testbed 13, the Authorization Server must accept the OAuth2 plugin registration based on a software statement, signed by Secure Dimensions GmbH. Two options exist how to make the software statement available: The first option is to provide it for download from the entity's web site and the second is to bundle the software statement with the binary build.

### 7.5.1. Software Statement for Download

Basically, the user must download the software statement in the first step and then store it locally. It is recommended but not required to verify the digital signature to make sure that the software statement is not corrupted.

The following figure illustrates the sequence of interactions between the QGIS application, the plugin as well as the user and the Authorization Server when using a software statement for

download.

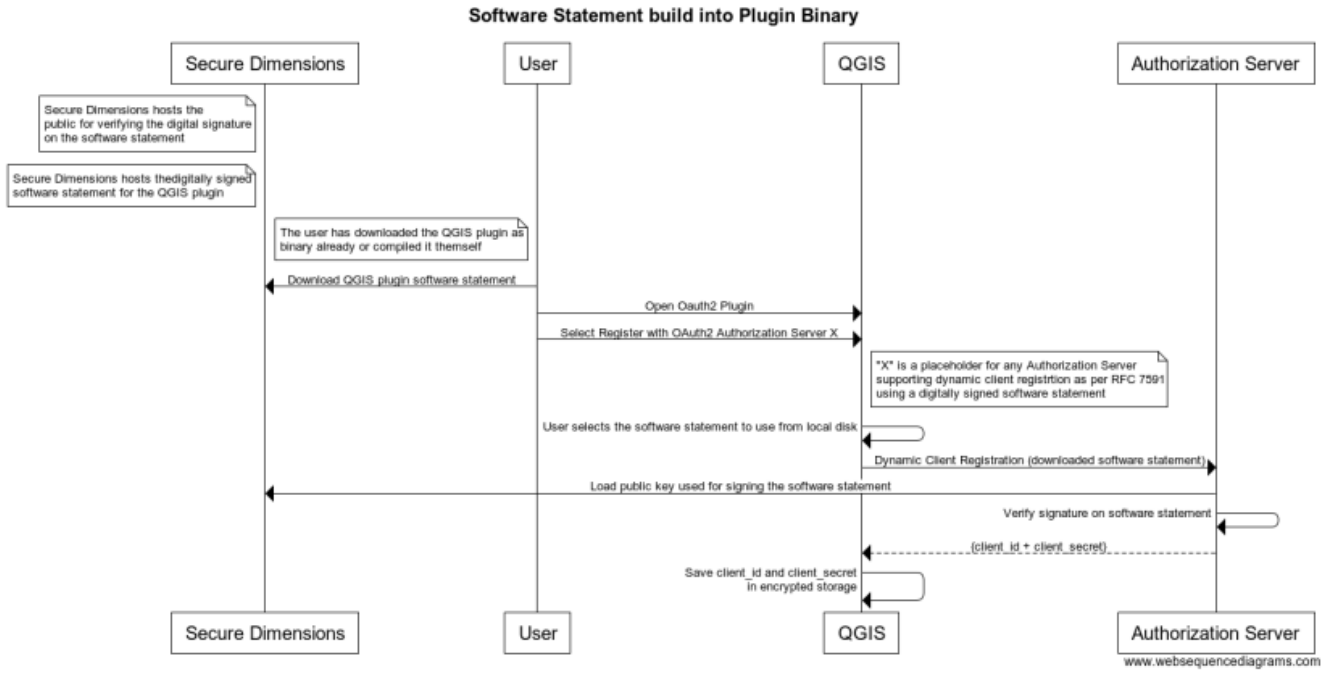


*Figure 5. OAuth2 plugin registration for TB13*

### 7.5.2. Software Statement built into Binary Bundle

The entity that builds the plugin binary, for TB13 that is Secure Dimensions, can include the software statement into the plugin binary. The user must then download and install the plugin as usual, but must not download the software statement in a separate step. For registration of the plugin with the different OAuth2 Authorization Server, the plugin will simply send the built-in software statement. It is recommended but not required that the user validates the hash value from the binary with the published hash value after downloading to make sure the plugin is not corrupt. For TB13, Secure Dimensions provided the QGIS OAuth2 plugin binary for download and published the correct hash value.

*Note*

```
The URL for downloading the plugin binary is https://as.tb13.secure-dimensions.de
```

The following figure illustrates the cause of interactions to use the build in software statement for registration of the plugin with an OAuth2 Authorization Server.

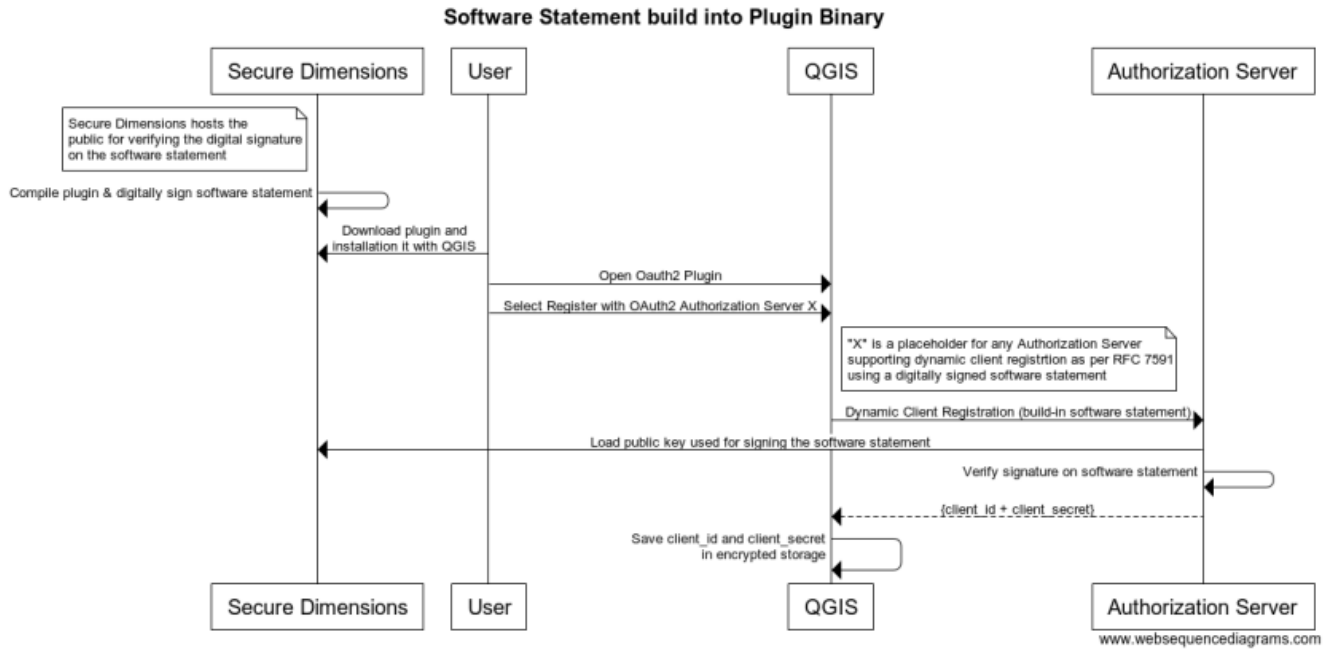


*Figure 6. OAuth2 plugin registration for TB13*

### 7.5.3. Manual registration

In cases where the OAuth2 Authorization Server does not support the Dynamic Client Registration via a digitally signed software statement, it must still be possible to register the plugin. The implementation supports this by asking the user to input the client_id and client_secret that was received from registering the plugin manually. The manual registration must follow the guidance from the Authorization Server that will be used. The following URLs are starting points for the Google and Facebook registration:

- Google: https://console.developers.google.com
- Facebook: https://developers.facebook.com

The following figure (Figure 7) displays the result from the Google registration:

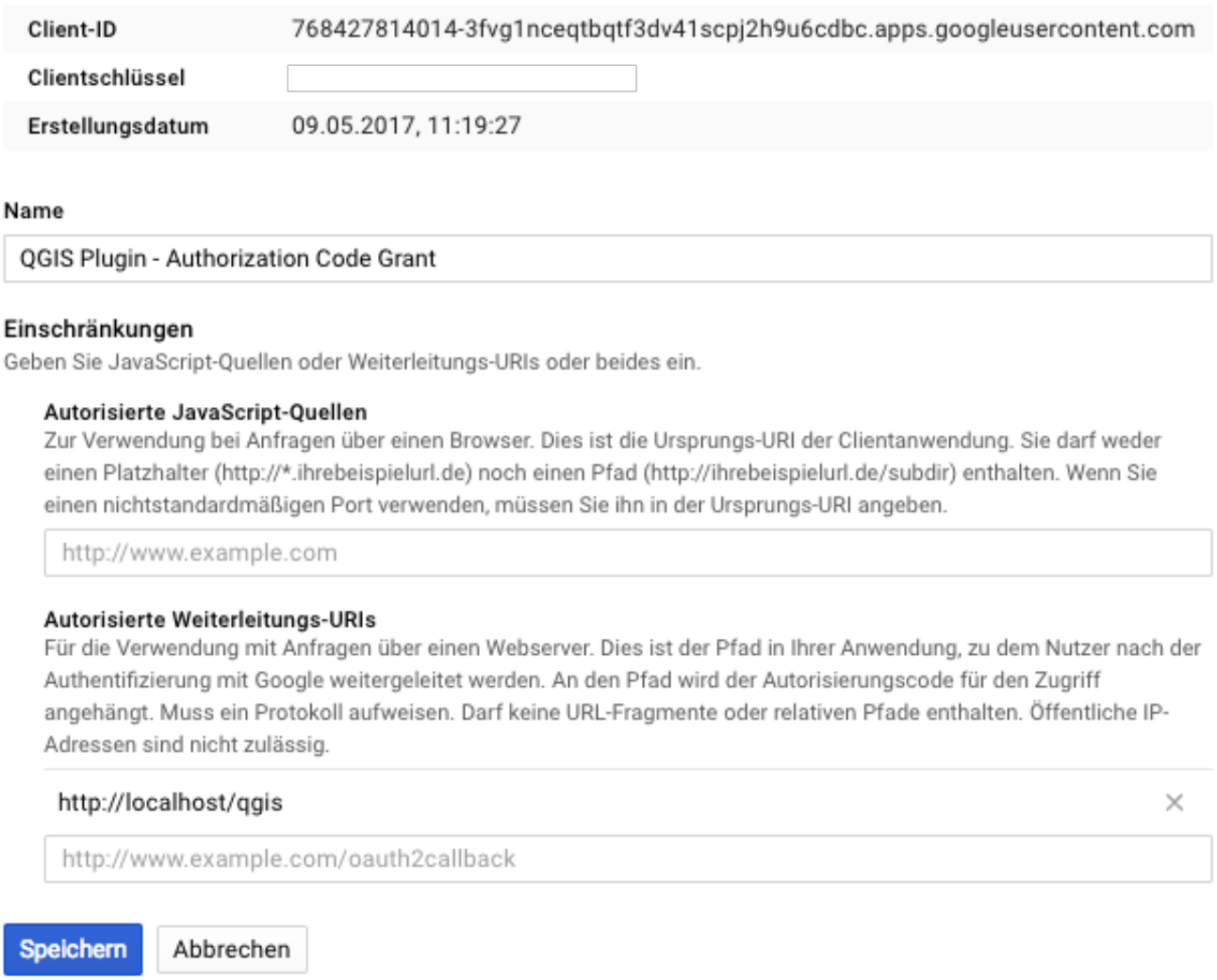


*Figure 7. OAuth2 plugin manual registration with Google*

The QGIS user must copy and paste the value of the "Client-ID" (client id) and the value of the "Clientschlüssel" (client secret) into the OAuth2 plugin configuration.

The following figure (Figure 8) displays the result from the Facebook registration:

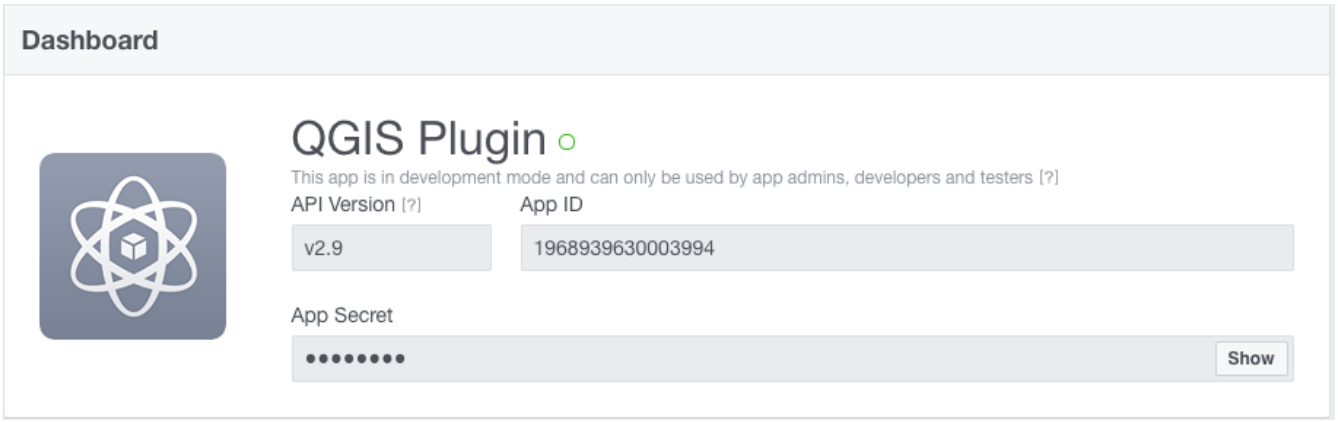


*Figure 8. OAuth2 plugin manual registration with Facebook*

The following sequence diagram in Figure 9 illustrates the functioning of the plugin and QGIS when selecting the manual configuration of the plugin to work with an Authorization Server, where the client credentials were received manually.

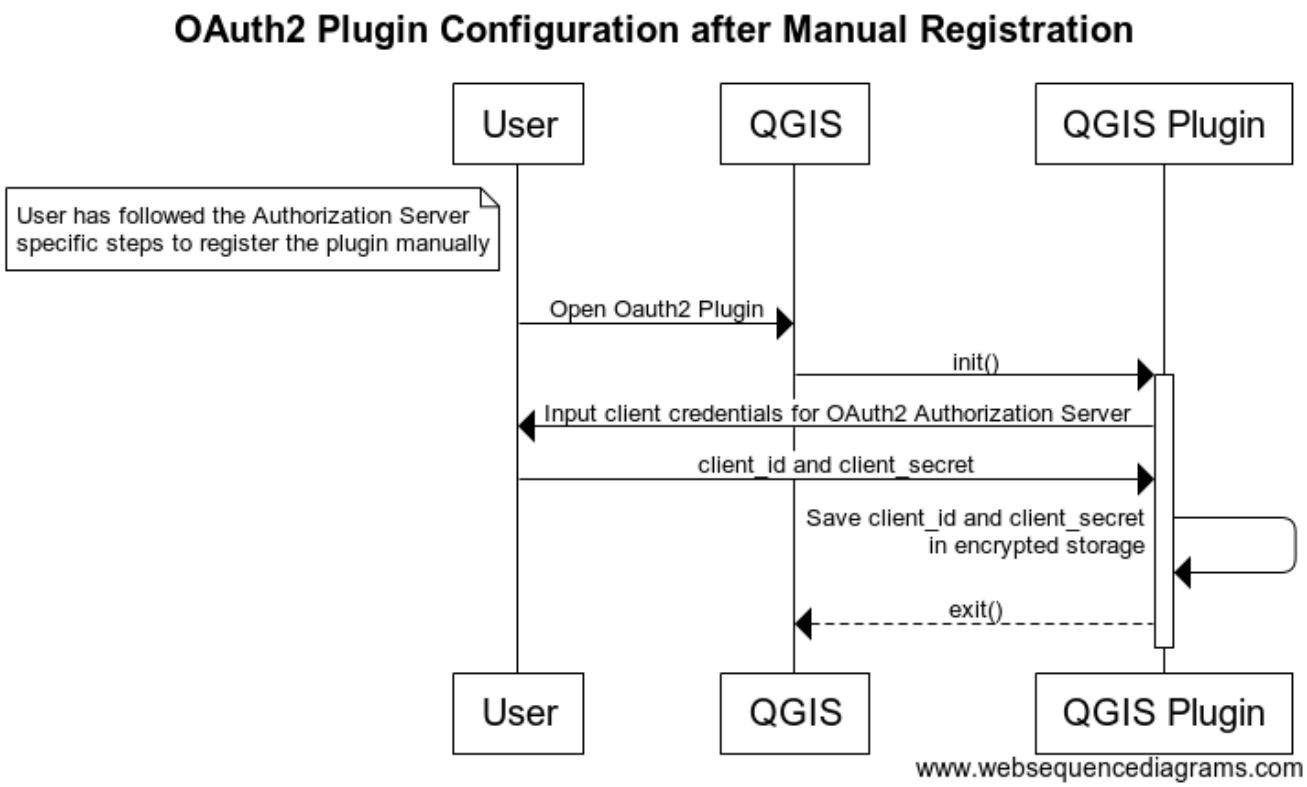


*Figure 9. OAuth2 plugin configuration after manual registration for TB13*

The user selects "Options/Authentication" from the QGIS menue. Then, QGIS opens the selected plugin via the init() method. When selecting OAuth2, the plugin opens a GUI asking the user to provide different details, depending on which grant type is selected by the user. Two major types of input are the client id and the client secret. How to get these application credentials was illustrated for different Authorization Servers in the figures Figure 7 and Figure 8.

## 7.6. Software Statements for Testbed 13

For Testbed 13, the QGIS OAuth2 Plugin can leverage the OAuth2 Authorization Code Grant and the Resource Owner Password Credentials Grant. In order to honor each grant type, different software statements are provided.

For the Authorization Code Grant, it is required that the authorization code is returned to the application via a redirect URI. As explained before, this is typically achieved by leveraging an open port on localhost. However, some computing environments may not allow this.

The Authorization Code flow can be split into two sequences to prevent the use of the open port and redirect to localhost. In this case, the Authorization Server does return the authorization code in the user-agent and the user must copy and paste the code into the QGIS plugin. In order to achieve that, the Authorization Server must provide a dedicated URL for just displaying the authorization code. For Testbed 13, the Secure Dimensions Authorization Server provides the following redirect URI for this purpose: https://as.tb13.secure-dimensions.de/oauth/display_code.php

Google's Authorization Server provides two specific redirect URIs that define a particular way of how the authorization code shall be returned to the user-agent and how the user-agent shall display the code (https://developers.google.com/api-client-library/python/auth/installed-app):

- urn:ietf:wg:oauth:2.0:oob: The code is returned in an html document where the title is equal to the authorization code. The page itself shall also display the code and ask the user to copy and paste the code.
- urn:ietf:wg:oauth:2.0:oob:auto: Like urn:ietf:wg:oauth:2.0:oob but close the window after display.

*Note*

```
The Google specific redirect URIs are not supported by the Authorization Server
provided by Secure Dimensions.
```

### 7.6.1. Software Statement for the Authorization Code Grant (display code)

For Testbed 13, the registration of the QGIS OAuth2 plugin for the Authorization Code Grant with no redirect to localhost is based on the following software statement:

```
{
  "iss": "www.secure-dimensions.de",
  "aud": "as.tb13.secure-dimensions.de",
  "software_id": "0e3eff22-5690-4306-aafb-8d4d0f5b967b",
  "software_version": "2.18.4x",
  "client_name": "QGIS OAuth2 Plugin",
  "client_uri": "https://qgis.org",
  "redirect_uris": ["https://as.tb13.secure-dimensions.de/oauth/display_code.php"],
  "token_endpoint_auth_method": "client_secret_post",
  "grant_types": ["authorization_code"],
  "response_types": ["code"],
  "logo_uri": "https://hub.qgis.org/attachments/4003/QGis_Logo.png",
  "scope": "openid beeld",
  "contacts": ["mailto:am@secure-dimensions.de"],
  "tos_uri": "https://as.tb13.secure-dimensions.de/qgis-tos.html",
  "policy_uri": "https://as.tb13.secure-dimensions.de/qgis-policy.html",
  "jwks_uri": "https://as.tb13.secure-dimensions.de/.well-known/jwks.json",
  "kid": "SDPublicKey"
}
```

This software statement can be used for dynamic client registration after converted to a signed JWT:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUyJ9.eyJpc3MiOiJ3d3cuc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJhd
WQiOiJhcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlIiwic29mdHdhcmVfaWQiOiIwZTNlZmYyMi01NjkwLTQ
zMDYtYWFmYi04ZDRkMGY1Yjk2N2IiLCJzb2Z0d2FyZV92ZXJzaW9uIjoiMi4xOC40eCIsImNsaWVudF9uYW1lI
joiUUdJUyBPQXV0aDIgUGx1Z2luIiwiY2xpZW50X3VyaSI6Imh0dHBzOi8vcWdpcy5vcmciLCJyZWRpcmVjdF9
1cmlzIjpbImh0dHBzOi8vYXMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS9vYXV0aC9kaXNwbGF5X2NvZGUuc
GhwIl0sInRva2VuX2VuZHBvaW50X2F1dGhfbWV0aG9kIjoiY2xpZW50X3NlY3JldF9wb3N0IiwiZ3JhbnRfdHl
wZXMiOlsiYXV0aG9yaXphdGlvbl9jb2RlIl0sInJlc3BvbnNlX3R5cGVzIjpbImNvZGUiXSwibG9nb191cmkiO
iJodHRwczovL2h1Yi5xZ2lzLm9yZy9hdHRhY2htZW50cy80MDAzL1FHaXNfTG9nby5wbmciLCJzY29wZSI6Im9
wZW5pZCBiZWVsZCIsImNvbnRhY3RzIjpbIm1haWx0bzphbUBzZWN1cmUtZGltZW5zaW9ucy5kZSJdLCJ0b3Nfd
XJpIjoiaHR0cHM6Ly9hcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlL3FnaXMtdG9zLmh0bWwiLCJwb2xpY3l
fdXJpIjoiaHR0cHM6Ly9hcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlL3FnaXMtcG9saWN5Lmh0bWwiLCJqd
2tzX3VyaSI6Imh0dHBzOi8vYXMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS8ud2VsbC1rbm93bi9qd2tzLmp
zb24iLCJraWQiOiJTRFB1YmxpY0tleSIsImlhdCI6MTQ5OTE1MDY3M30.M4ablN_BfmHjVPYZo_uVrCDYLsaal
hF_aRsS-
hppT9AJd8DAGZn_xYrsq8FIaKjZhivjKHtjbR8zyFee0HjU6iMP9KgC0N_jPn3o5L2n8IEl7oRJ5zW9V-
v2SqBdaXffm7TBx7v8KQ2J4uaoARYWgAkBcVVxKZWX9kLgQfaaoSu2Zk-
aoMNqzNU8u2UTADpoCwGjOd10ik0ZdQ6VC2czfAfYEmfYg4UYPQDafKjWWEXUhO9PXzD0piv_NJ9o2oDdc34KB
j8brZRDbDfuvIP5dxpaDh8PWhVE4Dd7dEsAzPXEFdXnLTGzQ_5zDj_CcUQqJ0SzkohALGhpPgy51Bjwsw
```

The actual request would look like this

```
POST /oauth/register.php HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: as.tb13.secure-dimensions.de

{
"software_statement":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUyJ9.eyJpc3MiOiJ3d3cuc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJh
dWQiOiJhcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlIiwic29mdHdhcmVfaWQiOiIwZTNlZmYyMi01NjkwLT
QzMDYtYWFmYi04ZDRkMGY1Yjk2N2IiLCJzb2Z0d2FyZV92ZXJzaW9uIjoiMi4xOC40eCIsImNsaWVudF9uYW1l
IjoiUUdJUyBPQXV0aDIgUGx1Z2luIiwiY2xpZW50X3VyaSI6Imh0dHBzOi8vcWdpcy5vcmciLCJyZWRpcmVjdF
91cmlzIjpbImh0dHBzOi8vYXMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS9vYXV0aC9kaXNwbGF5X2NvZGUu
cGhwIl0sInRva2VuX2VuZHBvaW50X2F1dGhfbWV0aG9kIjoiY2xpZW50X3NlY3JldF9wb3N0IiwiZ3JhbnRfdH
lwZXMiOlsiYXV0aG9yaXphdGlvbl9jb2RlIl0sInJlc3BvbnNlX3R5cGVzIjpbImNvZGUiXSwibG9nb191cmki
OiJodHRwczovL2h1Yi5xZ2lzLm9yZy9hdHRhY2htZW50cy80MDAzL1FHaXNfTG9nby5wbmciLCJzY29wZSI6Im
9wZW5pZCBiZWVsZCIsImNvbnRhY3RzIjpbIm1haWx0bzphbUBzZWN1cmUtZGltZW5zaW9ucy5kZSJdLCJ0b3Nf
dXJpIjoiaHR0cHM6Ly9hcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlL3FnaXMtdG9zLmh0bWwiLCJwb2xpY3
lfdXJpIjoiaHR0cHM6Ly9hcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlL3FnaXMtcG9saWN5Lmh0bWwiLCJq
d2tzX3VyaSI6Imh0dHBzOi8vYXMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS8ud2VsbC1rbm93bi9qd2tzLm
pzb24iLCJraWQiOiJTRFB1YmxpY0tleSIsImlhdCI6MTQ5OTE1MDY3M30.M4ablN_BfmHjVPYZo_uVrCDYLsaa
lhF_aRsS-
hppT9AJd8DAGZn_xYrsq8FIaKjZhivjKHtjbR8zyFee0HjU6iMP9KgC0N_jPn3o5L2n8IEl7oRJ5zW9V-
v2SqBdaXffm7TBx7v8KQ2J4uaoARYWgAkBcVVxKZWX9kLgQfaaoSu2Zk-
aoMNqzNU8u2UTADpoCwGjOd10ik0ZdQ6VC2czfAfYEmfYg4UYPQDafKjWWEXUhO9PXzD0piv_NJ9o2oDdc34KB
j8brZRDbDfuvIP5dxpaDh8PWhVE4Dd7dEsAzPXEFdXnLTGzQ_5zDj_CcUQqJ0SzkohALGhpPgy51Bjwsw"
}
```

As the result of the registration process, the plugin receives the client_id, client_secret which is saved with the redirect_uri inside the encrypted storage.

```
{
"client_id": "fd0ec452-bc2d-4786-8ed9-7befe9b6716c@www.secure-dimensions.de",
"client_secret": "c1b8d9d5dd2f1133e360164884ac55d7ba41649dc2a2ce7c4a2d060e6f9252c0"
}
```

### 7.6.2. Software Statement for the Authorization Code Grant (localhost redirect)

For Testbed 13, the registration of the QGIS OAuth2 plugin for the Authorization Code Grant with redirect to localhost is based on the following software statement:

```
{
  "iss": "www.secure-dimensions.de",
  "aud": "as.tb13.secure-dimensions.de",
  "software_id": "fd0ec452-bc2d-4786-8ed9-7befe9b6716c",
  "software_version": "2.18.4x",
  "client_name": "QGIS OAuth2 Plugin",
  "client_uri": "https://qgis.org",
  "redirect_uris": ["http://127.0.0.1/qgis", "http://[::1]/qgis"],
  "token_endpoint_auth_method": "client_secret_post",
  "grant_types": ["authorization_code"],
  "response_types": ["code"],
  "logo_uri": "https://hub.qgis.org/attachments/4003/QGis_Logo.png",
  "scope": "openid beeld",
  "contacts": ["mailto:am@secure-dimensions.de"],
  "tos_uri": "https://as.tb13.secure-dimensions.de/qgis-tos.html",
  "policy_uri": "https://as.tb13.secure-dimensions.de/qgis-policy.html",
  "jwks_uri": "https://as.tb13.secure-dimensions.de/.well-known/jwks.json",
  "kid": "SDPublicKey"
}
```

This software statement can be used for dynamic client registration after converted to a signed JWT:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUyJ9.eyJpc3MiOiJ3d3cuc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJhd
WQiOiJhcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlIiwic29mdHdhcmVfaWQiOiJmZDBlYzQ1Mi1iYzJkLTQ
3ODYtOGVkOS03YmVmZTliNjcxNmMiLCJzb2Z0d2FyZV92ZXJzaW9uIjoiMi4xOC40eCIsImNsaWVudF9uYW1lI
joiUUdJUyBPQXV0aDIgUGx1Z2luIiwiY2xpZW50X3VyaSI6Imh0dHBzOi8vcWdpcy5vcmciLCJyZWRpcmVjdF9
1cmlzIjpbImh0dHA6Ly8xMjcuMC4wLjEvcWdpcyIsImh0dHA6Ly9bOjoxXS9xZ2lzIl0sInRva2VuX2VuZHBva
W50X2F1dGhfbWV0aG9kIjoiY2xpZW50X3NlY3JldF9wb3N0IiwiZ3JhbnRfdHlwZXMiOlsiYXV0aG9yaXphdGl
vbl9jb2RlIl0sInJlc3BvbnNlX3R5cGVzIjpbImNvZGUiXSwibG9nb191cmkiOiJodHRwczovL2h1Yi5xZ2lzL
m9yZy9hdHRhY2htZW50cy80MDAzL1FHaXNfTG9nby5wbmciLCJzY29wZSI6Im9wZW5pZCBiZWVsZCIsImNvbnR
hY3RzIjpbIm1haWx0bzphbUBzZWN1cmUtZGltZW5zaW9ucy5kZSJdLCJ0b3NfdXJpIjoiaHR0cHM6Ly9hcy50Y
jEzLnNlY3VyZS1kaW1lbnNpb25zLmRlL3FnaXMtdG9zLmh0bWwiLCJwb2xpY3lfdXJpIjoiaHR0cHM6Ly9hcy5
0YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlL3FnaXMtcG9saWN5Lmh0bWwiLCJqd2tzX3VyaSI6Imh0dHBzOi8vY
XMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS8ud2VsbC1rbm93bi9qd2tzLmpzb24iLCJraWQiOiJTRFB1Ymx
pY0tleSIsImlhdCI6MTQ5OTE1MDY3OX0.K1vFNTpQWDgfPzpH5z_C3GRPSNszIUuh1v5n4_wM7PX8ypx4qvsxe
OADkHkosG3W0TtnMYL-
HHCvEEPvzf8TzxliQl_OptLu8wBzWb8efrVUzALyGG14n52llVHh7d0aIVGkPcseSbHEgWrg51jnRnz0osBU66
dsKKhe3joDJLrwkcuIhusJyTSeBMxj1-5W-
68L66dg8VM6NjsbYR9px5805TLyoVpv2liSBitx0yyRnNqBefAQYPqLcAMt3o3BlziH1r7EYiHppi-
TykO7wh7Q2UtoQ4Xo8MuWz089cXLQQhqmJpTwHHHu_aZm0DDrPUNFShfPGDcXBim-LCFziw
```


### 7.6.3. Registration for the Resource Owner Password Credentials Grant

For Testbed 13, the registration of the QGIS OAuth2 plugin for the Resource Owner Password Credentials Flow is based on the following software statement:


```
{
  "iss": "www.secure-dimensions.de",
  "aud": "as.tb13.secure-dimensions.de",
  "software_id": "5fffa83e-33de-ab93-ef34-44938dde",
  "software_version": "2.18.4x",
  "client_name": "QGIS OAuth2 Plugin",
  "client_uri": "https://qgis.org",
  "redirect_uris": [""],
  "token_endpoint_auth_method": "client_secret_basic",
  "grant_types": ["password"],
  "response_types": ["token"],
  "logo_uri": "https://hub.qgis.org/attachments/4003/QGis_Logo.png",
  "scope": "openid beeld",
  "contacts": ["mailto:am@secure-dimensions.de"],
  "tos_uri": "https://as.tb13.secure-dimensions.de/qgis-tos.html",
  "policy_uri": "https://as.tb13.secure-dimensions.de/qgis-policy.html",
  "jwks_uri": "https://as.tb13.secure-dimensions.de/.well-known/jwks.json",
  "kid": "SDPublicKey"
}
```


This software statement can be used for dynamic client registration after converted to a signed JWT:

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUyJ9.eyJpc3MiOiJ3d3cuc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJhd
WQiOiJhcy50YjEzLnNlY3VyZS1kaW1lbnNpb25zLmRlIiwic29mdHdhcmVfaWQiOiI1ZmZmYTgzZS0zM2RlLWF
iOTMtZWYzNC00NDkzOGRkZSIsInNvZnR3YXJlX3ZlcnNpb24iOiIyLjE4LjR4IiwiY2xpZW50X25hbWUiOiJRR
0lTIE9BdXRoMiBQbHVnaW4iLCJjbGllbnRfdXJpIjoiaHR0cHM6Ly9xZ2lzLm9yZyIsInJlZGlyZWN0X3VyaXM
iOlsiIl0sInRva2VuX2VuZHBvaW50X2F1dGhfbWV0aG9kIjoiY2xpZW50X3NlY3JldF9iYXNpYyIsImdyYW50X
3R5cGVzIjpbInBhc3N3b3JkIl0sInJlc3BvbnNlX3R5cGVzIjpbInRva2VuIl0sImxvZ29fdXJpIjoiaHR0cHM
6Ly9odWIucWdpcy5vcmcvYXR0YWNobWVudHMvNDAwMy9RR2lzX0xvZ28ucG5nIiwic2NvcGUiOiJvcGVuaWQgY
mVlbGQiLCJjb250YWN0cyI6WyJtYWlsdG86YW1Ac2VjdXJlLWRpbWVuc2lvbnMuZGUiXSwidG9zX3VyaSI6Imh
0dHBzOi8vYXMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS9xZ2lzLXRvcy5odG1sIiwicG9saWN5X3VyaSI6I
mh0dHBzOi8vYXMudGIxMy5zZWN1cmUtZGltZW5zaW9ucy5kZS9xZ2lzLXBvbGljeS5odG1sIiwiandrc191cmk
iOiJodHRwczovL2FzLnRiMTMuc2VjdXJlLWRpbWVuc2lvbnMuZGUvLndlbGwta25qc29vdnJvandrcy5qc29uIiwia
2lkIjoiU0RQdWJsaWNLZXkiLCJpYXQiOjE0OTkxNTA2NzZ9.Z47HNZ4Pvr30KQGAfBMBOQTQswotOqQsIb3ijn
_p8lRjWiu7mCk_3c-
YKSrhfheeMiYLXBN0SJZaxGwUjTOTGIv_YD6Y6raYxDy0mgWYFhDvAaofFmGsgCl6lhEJrYcHTHaGMgbPMulLL
GZvpASpNVbKhYfrze7hGXC1GevLkLF9pEJ4_wB9sqSVy6Q9SFTCvRGZGwddgXFhHzauOBcrVSDtCAKwGYlV9hf
afl32Nreizk8-ZyRVGm_O9171wPhtnpK62WOGhu6MbtWpc6miPqfRgHQ7UzyVO2OL-8OA7Hgu-e2jTrGxLM-
VzZLK5kK7n-nydrpMiKbVi0giclhstg
```

As the result of the registration process, the plugin receives the client_id and the client_secret which is stored with the redirect_uri inside the encrypted keystore.

The Authorization Server can use the following public key that is hosted at the jwks_uri https://as.tb13.secure-dimensions.de/jwks.json:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAy795JBHdeyI65zALG2rA
l4JRcCnWpued87fRoJxTknAnTqj/veFi/rz1EGAnbmUj04UndG5ILu+An95mZPNj
j28OqJ4UkUEUIxq5fAV12oA0mUYdjZqAblM/sbCFZ7FplEe6h98DRG4/sxwwTbas
cVPvH/XNjyBjve7iUqUnJJT8iZTI1cQBszciwa/C+MZwlHwvLjCN6CGBPQ2rzFsJ
ZPmJWtmi6vTL863sRdjlCTjN5WfctaiijI0ooLDgCYmuYB6uYf6MOBebKTaif+Zq
G+aaNYTriVinBXkaBW6THE56aKNfZM2e2+O9C+42qqCqF7GvMwSsFxK29YiaoGe6
zwIDAQAB
-----END PUBLIC KEY-----
```

## 7.7. The QGIS Authentication Framework

Since QGIS version 2.14, an authentication framework provides an API to configure authentication options for OGC Web Services connectors. This framework already provides authentication methods like BASIC and DIGEST as well as X.509 client certificate authentication.

For the purpose of implementing plugins for additional authentication methods, it is important to understand the class structure. The following figure illustrates in a simplified way the relevant classes and their relationships.

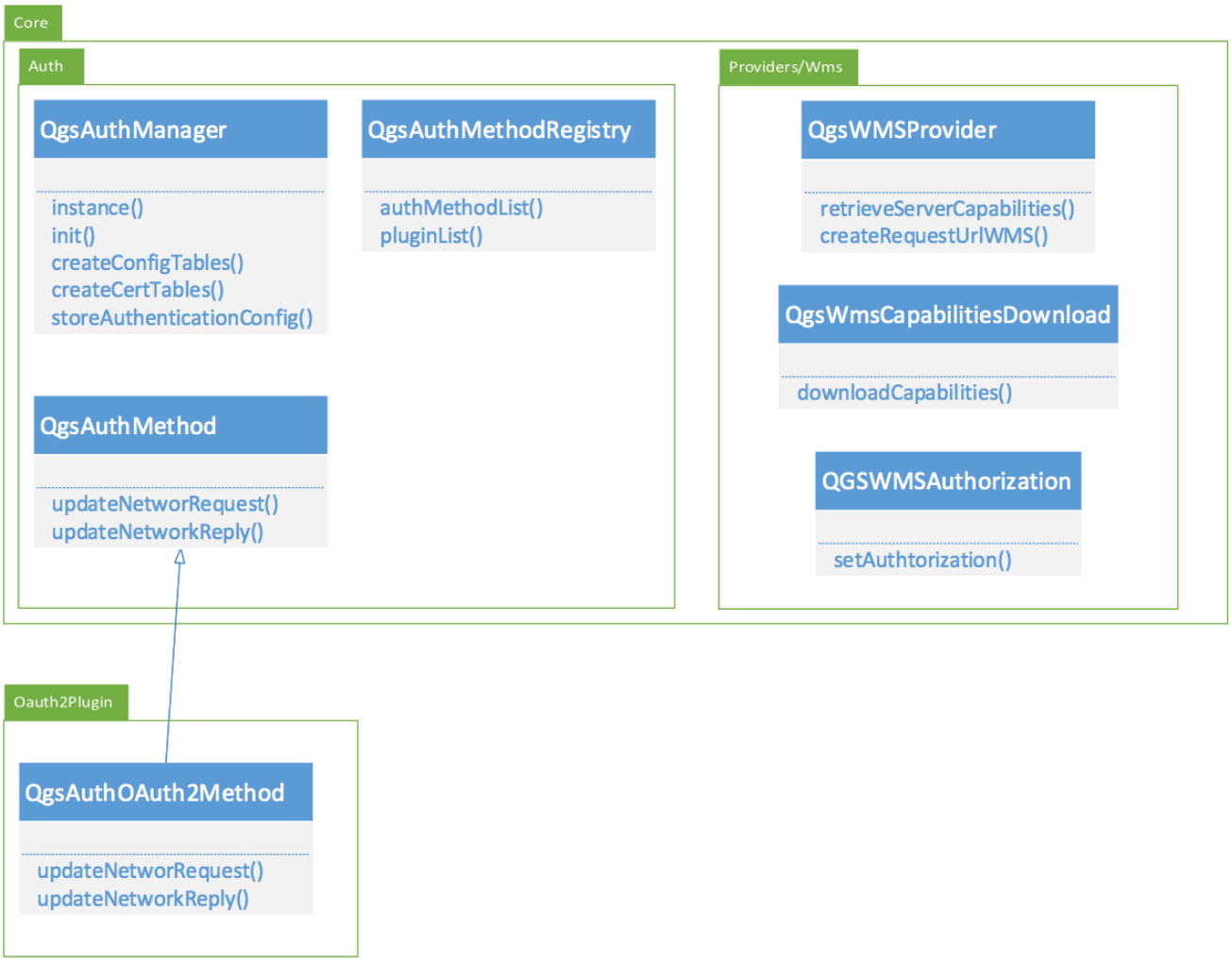


*Figure 10. QGIS Authentication Framework Class Diagram*

The following figure (Figure 11) illustrates, in a simplified way, the interactions of the authentication framework within the OGC Web Service adapters implemented in the QGIS core.

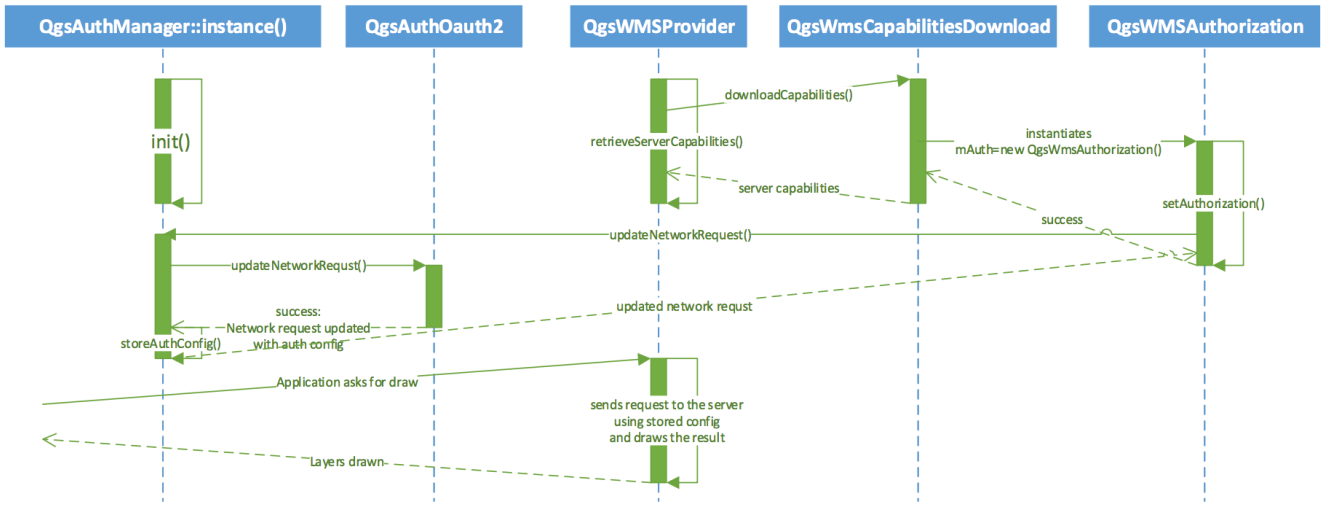


*Figure 11. QGIS Authentication Framework Sequence Diagram*

# Chapter 8. SAML2 Plugin

| | |
|---|---|
| **NOTE** | *Note*<br>This section contains the contributions from Secure Dimensions regarding the QGIS plugin for SAML2. |

This section documents the implementation of the SAML2 plugin for QGIS and the fit for purpose discussion of SAML2 profiles to be used for open source desktop applications.

## 8.1. The Trust Principle for SAML2

The trust in SAML is established between entities using SAML metadata. It is therefore not required that the actual application - the QGIS plugin - is trusted. The plugin must implement the SAML2 handshake to broker communication between the trusted entities (i) Service Provider and (ii) Identity Provider.

## 8.2. The SAML2 Plugin Implementation

The actual implementation of the plugin to support SAML2 based authentication is quite straight forward. The plugin must implement the SAML2 Profile "Enhanced Client Proxy" as described in the OASIS standard "*Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*".

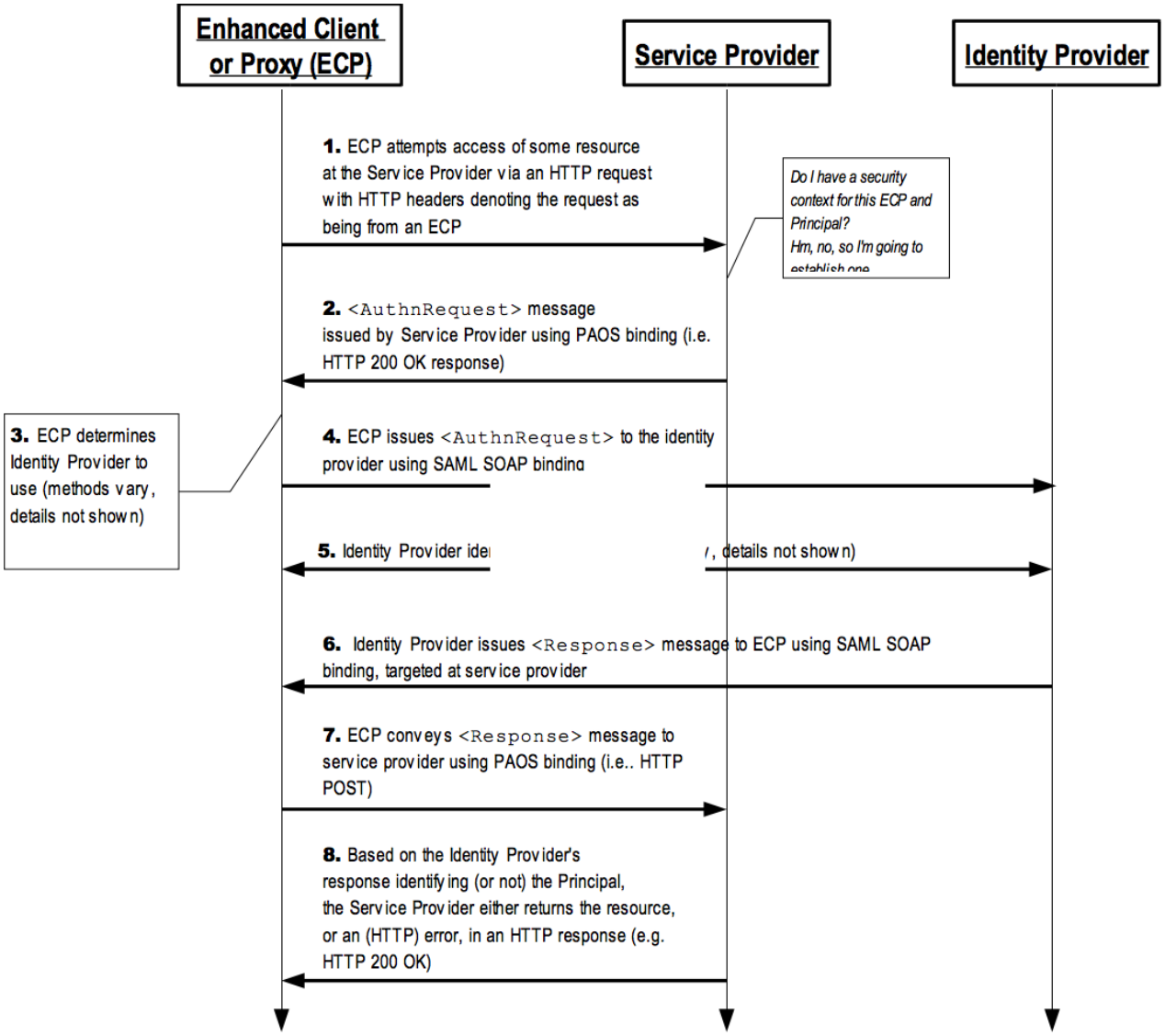


*Figure 12. SAML2 ECP*

In addition to supporting the ECP protocol, the plugin must also ensure that any subsequent request of QGIS providers (WMS, WFS, etc.) send the session cookie that was created as a result of a successful ECP interaction. This can be implemented by providing hooks into the OGC Web Service Providers as supported by the QGIS Authentication Framework implemented since v 2.14.

The following sequence diagram illustrates the basic interactions between the QGIS application (OWS Provider), the Authentication Plugin and the protected OWS service.

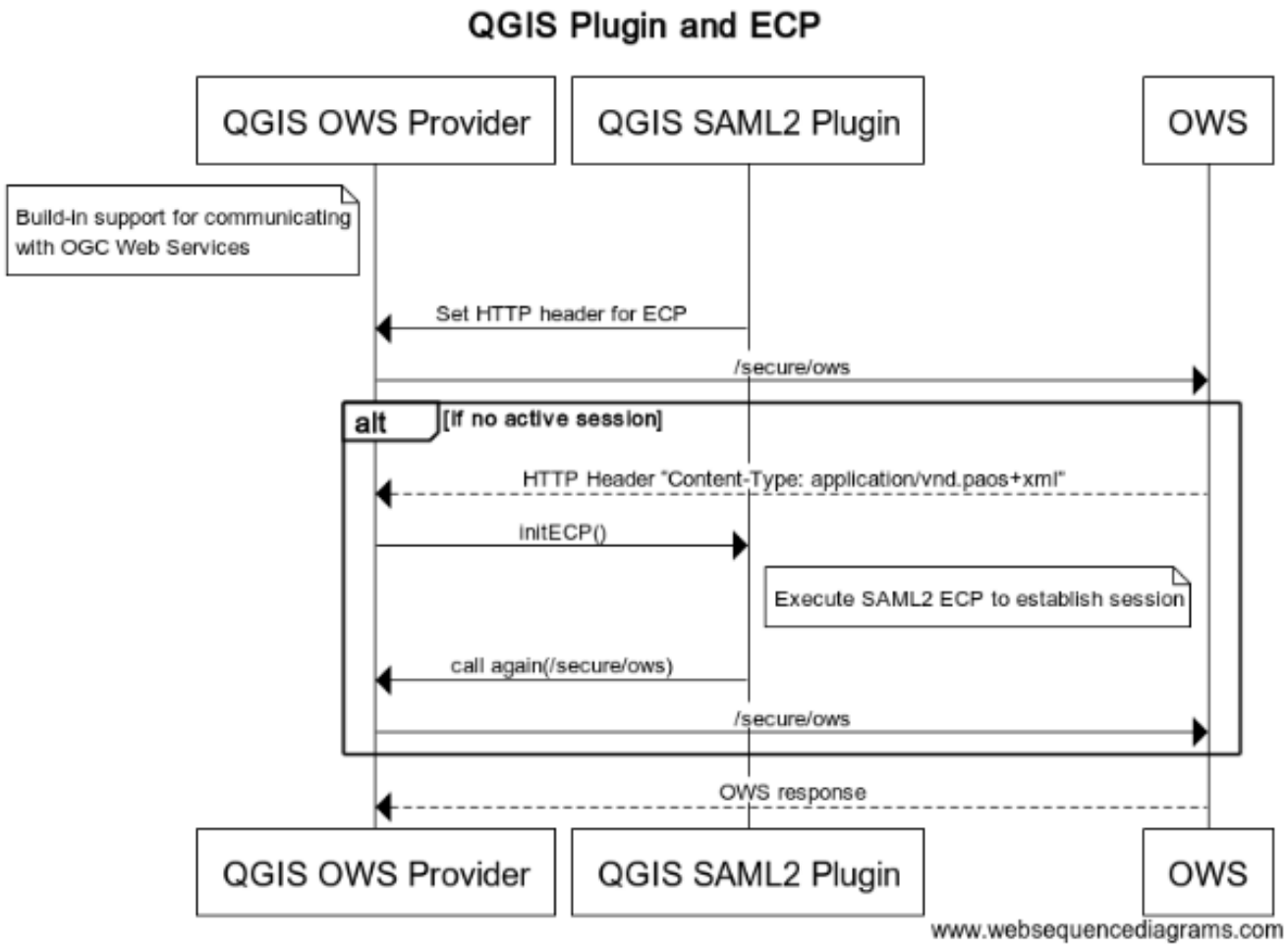


*Figure 13. Basic interactions for SAML2 ECP*

### 8.2.1. Identity Provider Discovery

The SAML2 ECP protocol does exclude the IdP discovery. Basically, two options exist how to implement this function:

- The first option would be that the user can load a list of IdPs as a kind of a custom configuration file. In this scenario, the developer must define a structure of the configuration file.
- The second option could be based on the list of IdPs returned by the Service Provider. But it is optional for the SP to return such a "hint".
- The third option would be that the plugin loads the SAML2 federation metadata that includes the IdP, which is to be used for login.

We implement the second option as it is based on parsing already existing information - each IdP must provide SAML2 metadata. And the structure is defined in the SAML2 Metadata specification.

The following pseudo code can be used for resolving IdPs and required information by parsing the SAML2 Metadata:

- Find “IdPSSODescriptor” element
- Find “Extensions” element
- Find “AuthnContextClassRef” element

    - Check values to contain an authentication class referring to Basic Authentication or X.509 Client Certificate
- Find “SingleSignOnService” element
- Find attribute “Binding” and check value to be “urn:oasis:names:tc:SAML:2.0:bindings:SOAP”
- Use URL value where to send AuthnResponse

Create entry in the internal IdP list for each qualifying IdP and keep the select IdP as required by the SAML2 ECP protocol.

### 8.2.2. Possible Authentication Methods

According to the SAML2 ECP protocol, the duty of the client implementing the profile is to relay XML SOAP messages between the SP and the IdP. At the point, where the client sends the AuthnResponse received from the SP to the IdP (4. in figure "SAML2 ECP") or during an extra instruction (5. in figure "SAML2 ECP"), the IdP might request the user to login. According to the ECP protocol, the actual authentication step(s) required are outside the scope of the ECP protocol.

Which authentication methods can be implemented in the client highly depend on the options provided by the programming language of the client. For the QGIS plugin, it would be possible to leverage a QTWebView to handle HTML page based login methods like username/password POST as it is the standard login with the Shibboleth IdP.

The authentication method that can always be implemented is HTTP Basic Authentication, as it does not require any additional HTML processing capabilities in the client. But this limitation in the implementation introduces implications to the IdP, as described in the section below.

### 8.2.3. ECP Implications to IdPs

The implementation of the ECP in the client requires that also IdPs support the profile. This is done via an IdP specific endpoint with SOAP binding. Therefore the part of the client implementation concerned with the IdP selection must verify that the IdP exposes something like the following profile element: <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" Location="https://idp.tb13.secure-dimensions.de/idp/profile/SAML2/SOAP/ECP"/> in its SAML metadata.

To support most client implementations, it is strongly recommended to protect this endpoint with HTTP Basic Authentication only. This guarantees maximum interoperability with all clients, including scripts and other programs that have no or a limited GUI.

### 8.2.4. Single-Sign-On

Unlike other SAML2 profiles, the SAML2 ECP does not guarantee Single-Sign-On (SSO) by default based on the protocol itself. It is up to the developer of the plugin to support SSO. Basically, the SSO support requires that the user must only select an IdP for the first connection to a protected endpoint at any Service Provider; the execution of protected services hosted by other Service Providers would resolve into an automatic session creation. In order to achieve this, the SAML2 plugin must "remember" the user’s IdP selection and leverage that when creating new sessions with Service Providers.

With the QGIS plugin, this can be supported quite easily, because the Authentication Framework supports configuration of an authentication method for each OGC Web Service. With this configuration, the IdP can be saved.

### 8.2.5. Session Lifetime

With SAML2, the session is managed and stored at the server side. It is therefore possible, that a request to a protected endpoint "all of a sudden" behaves different, because the session is expired.

What happens when the session is expired depends on the default behavior of the SAML Service Provider. Typically, it initiates the SAML2 Web Browser Profile which causes incompatibility with the plugin, as it does not support this profile. It is therefore important to ensure that the Service Provider always leverages the ECP protocol. This can be guaranteed by the plugin only if all requests to a protected endpoint carry the HTTP headers that indicate the use of ECP. The client can verify an expired session by the content-type returned by the Service Provider. In case the session is inactive, it will return content type "*application/vnd.paos+xml*". If the session is active, the Service Provider will return the content type as requested. For expired sessions as well as for active sessions, the HTTP status code is always 200.

# Chapter 9. OAuth-enabled Web Processing Service

| | |
|---|---|
| **NOTE** | *Note*<br>This section contains the contributions from 52°North regarding the OAuth enabled WPS. Some general introductions about OAuth may be moved to the standards section that also contains an introduction to OAuth. |

As illustrated above, the OAuth2 specification standardizes a framework for delegating access rights from users to applications. As such it is in particular useful for Web Processing Services that usually need to retrieve input datasets that may be access restricted.

In general, the WPS may take two OAuth roles:

- **WPS as OAuth client**: In this scenario, the WPS acts as an OAuth client that needs to retrieve resources owned by a user from an OAuth resource server. By means of OAuth, the user is able to authorize the WPS to access his resources without passing the credentials to the WPS.
- **WPS as OAuth resource server**: Being a resource server, the WPS processes are the resources owned by the user and other clients may request access to the protected process resources.

The core idea of OAuth is that authorization for an application to access a user's resource is realized by passing an access token that is issued by an OAuth authorization server. The server is mediating between the user and the client application requesting access. The abstract flow is shown in the figure below.

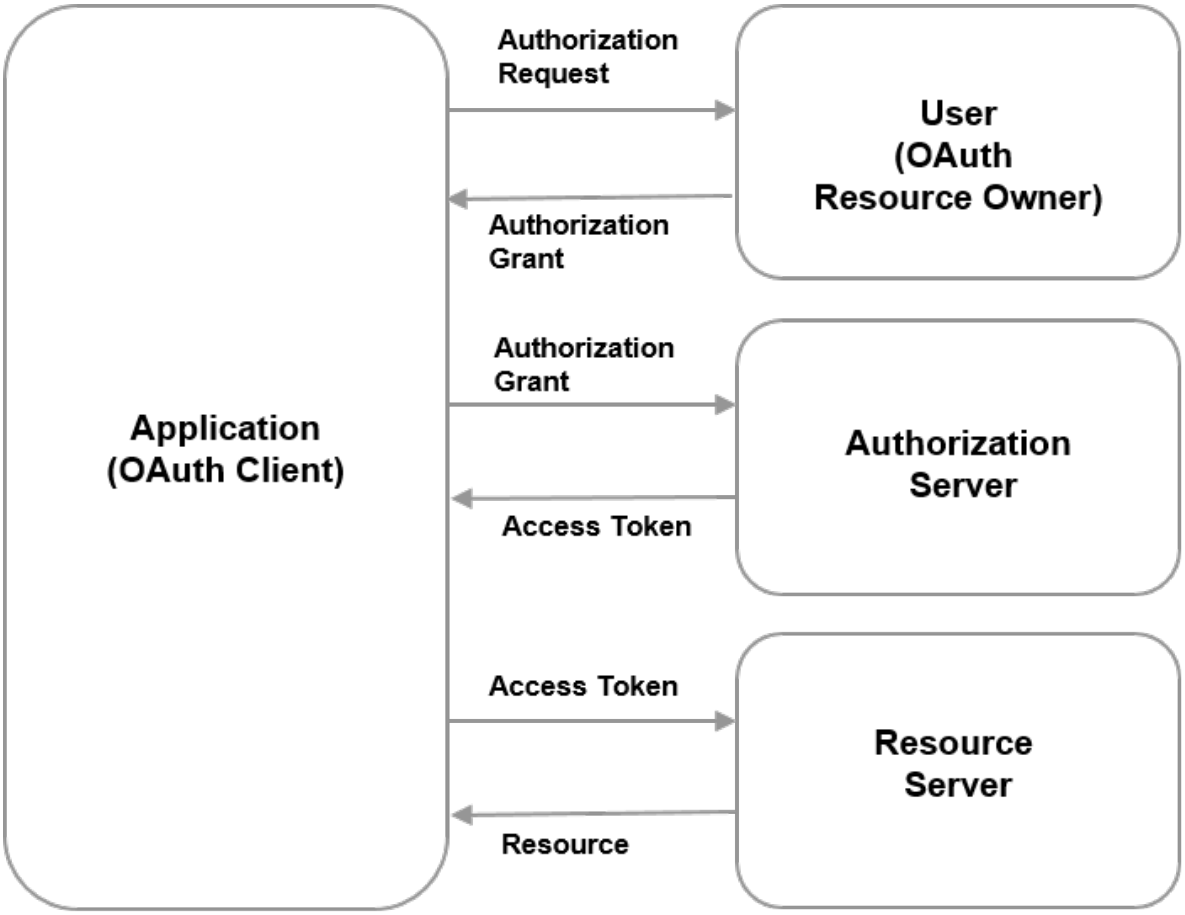


*Figure 14. Abstract OAuth authorization flow.*

At first the client application needs to request authorization from the user. As will be shown later, this request is usually done through redirects between the user agent (usually a Web browser), client application and the authorization server, where the authorization server authenticates the user and issues the authorization grant. Once the user permits access by providing his credentials in a Web form, an authorization grant is issued. This authorization grant is used by the client application to retrieve the actual access token from the authorization server that is needed to retrieve the protected resource from the resource server.

Different variations of this authorization flow are specified in the OAuth 2 specification depending on the client's capabilities for storing confidential information and the time when the actual authorization through the user happens:

- **Authorization Code Grant Flow**: This is the complete flow as described above, where the authorization server acts as an intermediary between the client and resource owner. The client directs the resource owner to an authorization server, which in turn directs the resource owner back to the client with the authorization code. Thereby, the resource owner only authenticates with the authorization server. The user credentials are never shared with the client. This flow is needed for implementing a WPS as OAuth client.
- **Implicit Grant Flow**: This is a simplified version of the authorization code flow optimized for clients running in a browser with scripting languages like Java script. Since the client cannot store confidential information, the client is issued an access token directly instead of an authorization grant. The grant type is called implicit, since no intermediate credentials are issued. The drawback of this method is that the authorization server does not authenticate the client, though the client's identity may be verified via the redirect URL that is used for delivery

of the access token and needs to be registered at the authorization server with the client beforehand. Since the WPS is able to store the authorization grant, consider the authorization code grant flow as a better option for implementing the WPS as OAuth client.

- **Resource Owner Password Credentials Flow**: In this case, the resource owner's password credentials are directly used as authorization grant to obtain an access token. This would require a high degree of trust between the client and the resource owner.
- **Client Credentials Flow**: In this case, the client is either also a resource owner or, as we consider for the WPS as OAuth client, the authorization has been previously arranged with the authorization server. For this purpose, some authorization server provider, offer for example the option to arrange such authorizations on a dashboard running in a browser.

To summarize, we consider two flows as options for implementing the WPS as OAuth client: The Authorization Code Flow and, in case authorization has been arranged beforehand, the Client Credentials Flow.

## 9.1. WPS as OAuth Client

Acting as an OAuth client, the WPS needs access to resources that are provided by OGC W*S services like the WCS. The interaction between the components is shown in the figure below.

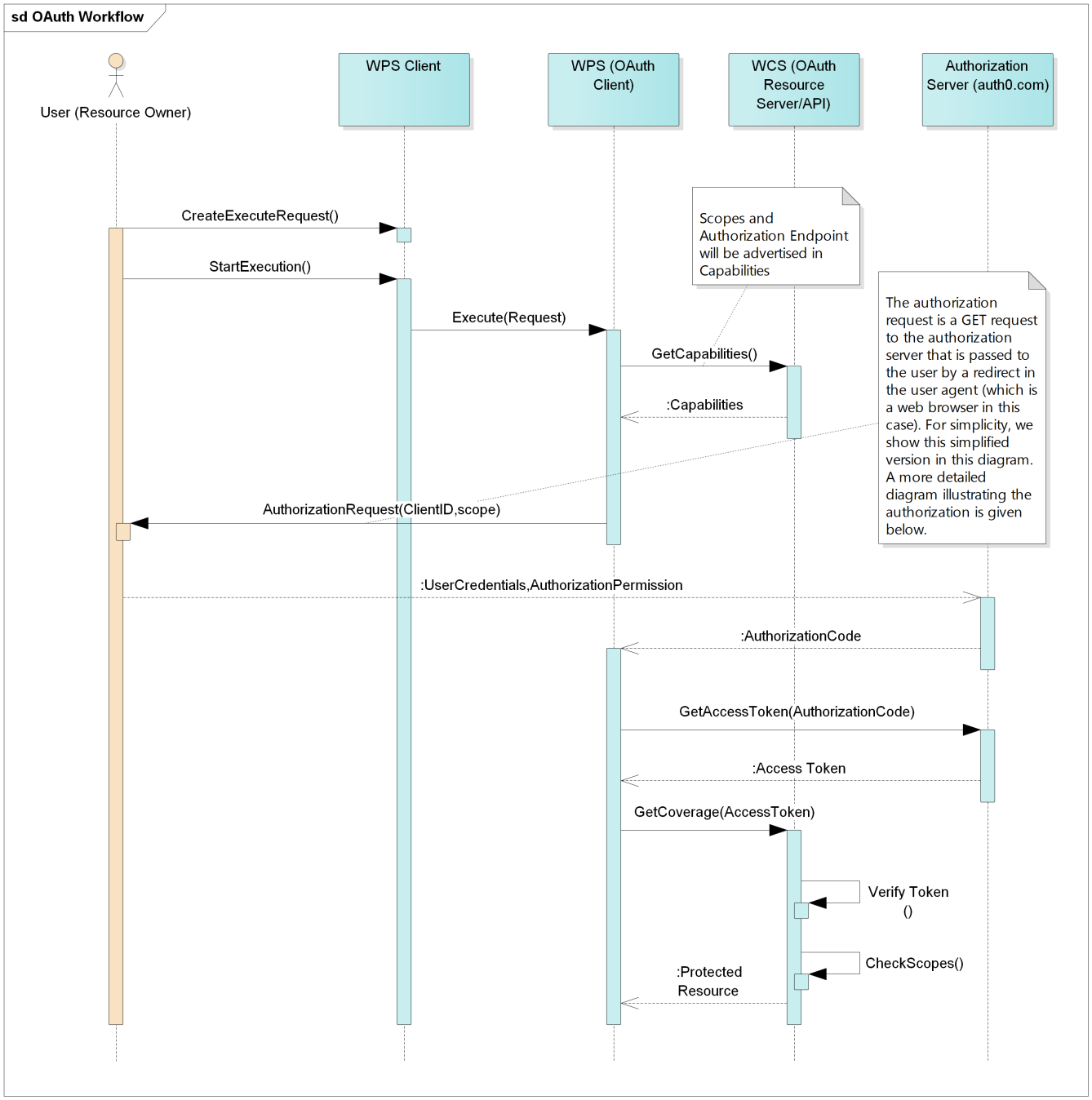


*Figure 15. Sequence diagram illustrating the WPS as OAuth Client.*

Both, the WPS and the WCS need to be registered at the authorization server beforehand. The mapping to access constraints is defined by so called scopes in OAuth. These are previously defined by the user when registering the WCS as resource server at the authorization server. In the Testbed 13 workflow scenario, where the OAuth WPS is utilized, Auth0 (https://auth0.com/) is used as authorization server. Auth0 also provides a powerful Web-based user interface for registering components as clients or resource servers, for defining scopes and also for granting authorization before running WPS processes in the Client Credentials Flow.

To trigger the execution of a WPS process, the WPS client sends an Execute request to the WPS containing inputs that reference coverage data provided by a WCS. The WCS acts as resource server, the owner of the referenced datasets is the user that executes the WPS process. Once the WPS retrieves the Capabilities from the WCS, it encounters that the GetCoverage operation is secured by OAuth. Therefore, a link to the scope listing and the URL of the authorization server are given in the Capabilities as constraints in the operations metadata following the proposal of the

currently developed OWS Common Security Extension. With this information, the WPS is able to compose an authorization request, i.e. a URL that is redirected to the user agent (usually a Web browser) containing the client ID and scopes for the requested resource. If the user invokes the request, the authorization server generates a form for providing the user credentials and authorizing the requesting WPS to access the resources. Once the user has authenticated and confirmed authorization the authorization code is redirected back to the WPS and the WPS is then able to retrieve an access token from the authorization server that allows to retrieve the coverage dataset. Since the WCS is acting as a resource server, it needs to verify the access token and check the scopes before returning the protected resources.

### 9.1.1. Authorization Code Flow vs. Client Credentials Flow

Since the WPS client is not tightly coupled with the WPS, how can the user be redirected to the authorization server for granting authorization? It is proposed to utilize the asynchronous execution mode of the WPS for this purpose as shown in the figure below.

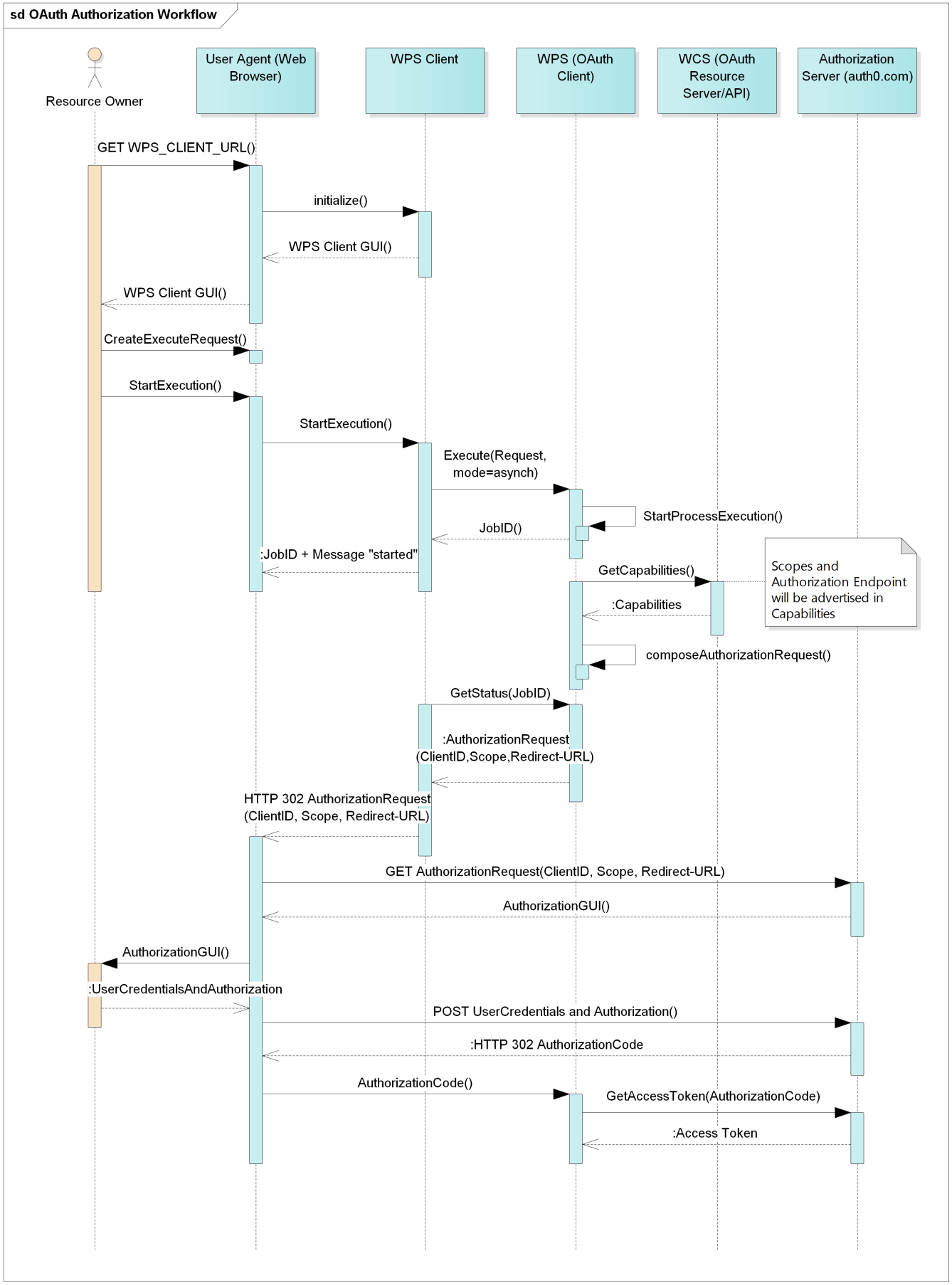


*Figure 16. Sequence diagram illustrating the WPS OAuth client implementation of the Authorization Code Flow.*

The user that executes the WPS process is also resource owner of the coverage datasets that are passed as inputs to the WPS. The WPS needs to be authorized to access the coverage datasets from

the WCS. The WPS Client is implemented as a JavaScript application running in the user agent (browser). The WPS provides a redirect-URL to which the authorization grant needs to be sent. When executing a WPS process in asynchronous mode, the WPS at first returns a job ID to the WPS client. This job ID is used for continuously polling the status from the WPS. Now, if the WPS needs to redirect the user to the authorization server for the authorization request, it can utilize the status response. Instead of returning the status information, it returns a redirect to the authorization server, whereas the URL to the authorization server represents the authorization request. The authorization server returns a GUI, i.e. an HTML form for providing the user credentials and confirming the authorization. The credentials and authorization is posted back to the authorization server that in turn redirects the authorization grant back to the redirect-URL of the WPS. The advantage of this approach is that we are relying upon the well-standardized asynchronous communication pattern for WPS, though it has to be noted that, in a strict sense, return of an HTTP 302 instead of a status response also does not conform with the current WPS specification and this behavior would need to be specified in an extension.

As can be seen above, the Authorization Code Flow allows for an ad-hoc authorization when executing a WPS process. As an alternative the Client credentials flow may be used in case authorization can be granted a priori by using external functionality of the authorization server like the dashboard provide by Auth0. The interaction between the different components is shown in the diagram below.

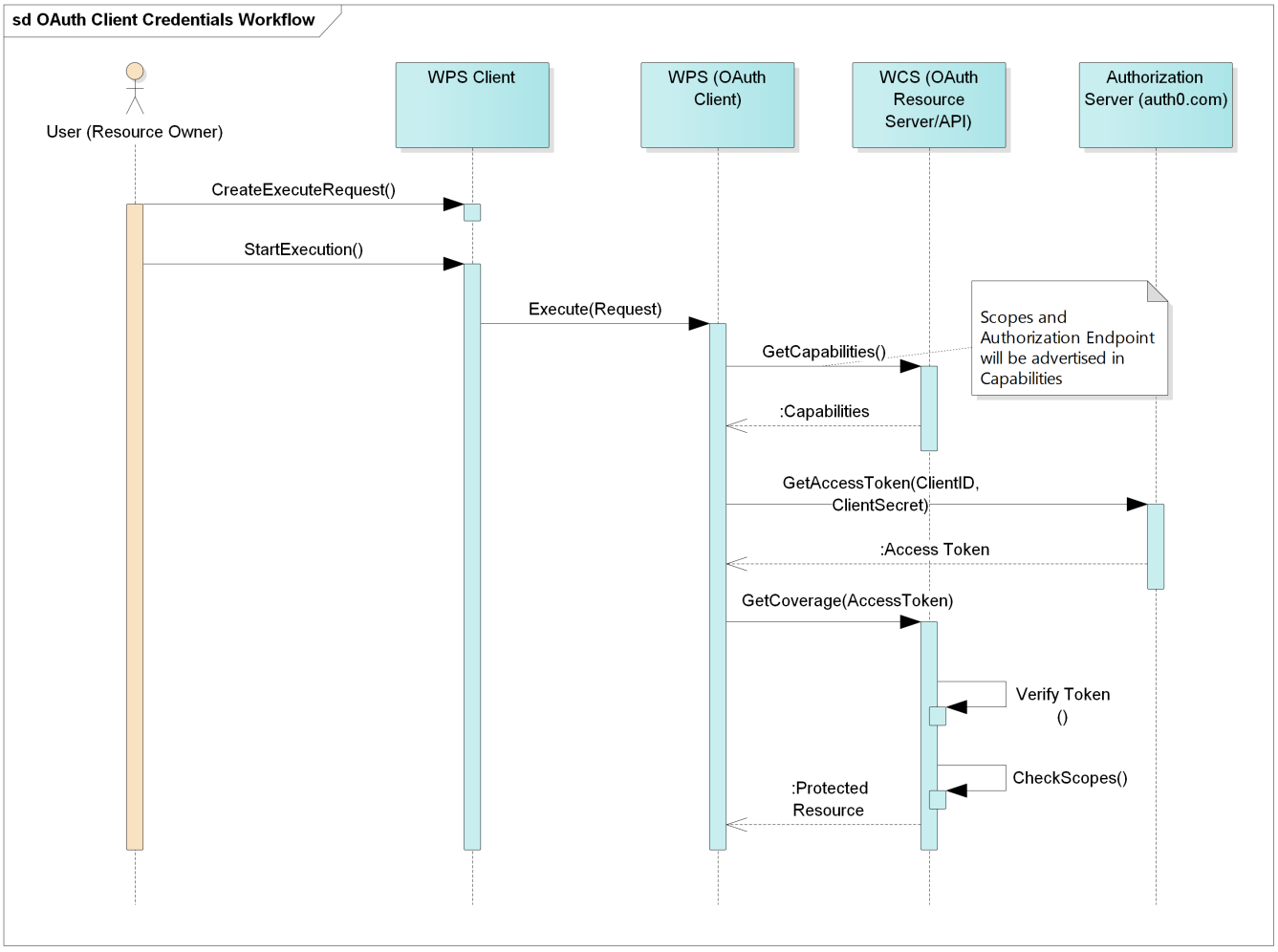


*Figure 17. Sequence diagram illustrating the WPS OAuth client implementation of the Authorization Code Flow.*

As can be seen the Client Credentials is much simpler and requires less communication between

the different components. The only step needed is that the WPS retrieves the access token by passing his client credentials to the authorization server. In case the user knows which WPS is used this would be the preferred option to implement, since both communication as well as implementation overhead is much smaller than utilizing the Authorization Code Flow. However, in case the WPS is not known to the user and a priori authorization is not possible, the authorization code flow would be needed. It is important to note that -as a pre-requisite- it is still assumed that the WPS is registered with a redirect-URL at the authorization server beforehand.

## 9.2. WPS as OAuth Resource Server

If the WPS is an OAuth Resource Server, it is offering its service operations and processes as resources that are owned by users who may provide access for clients or not.

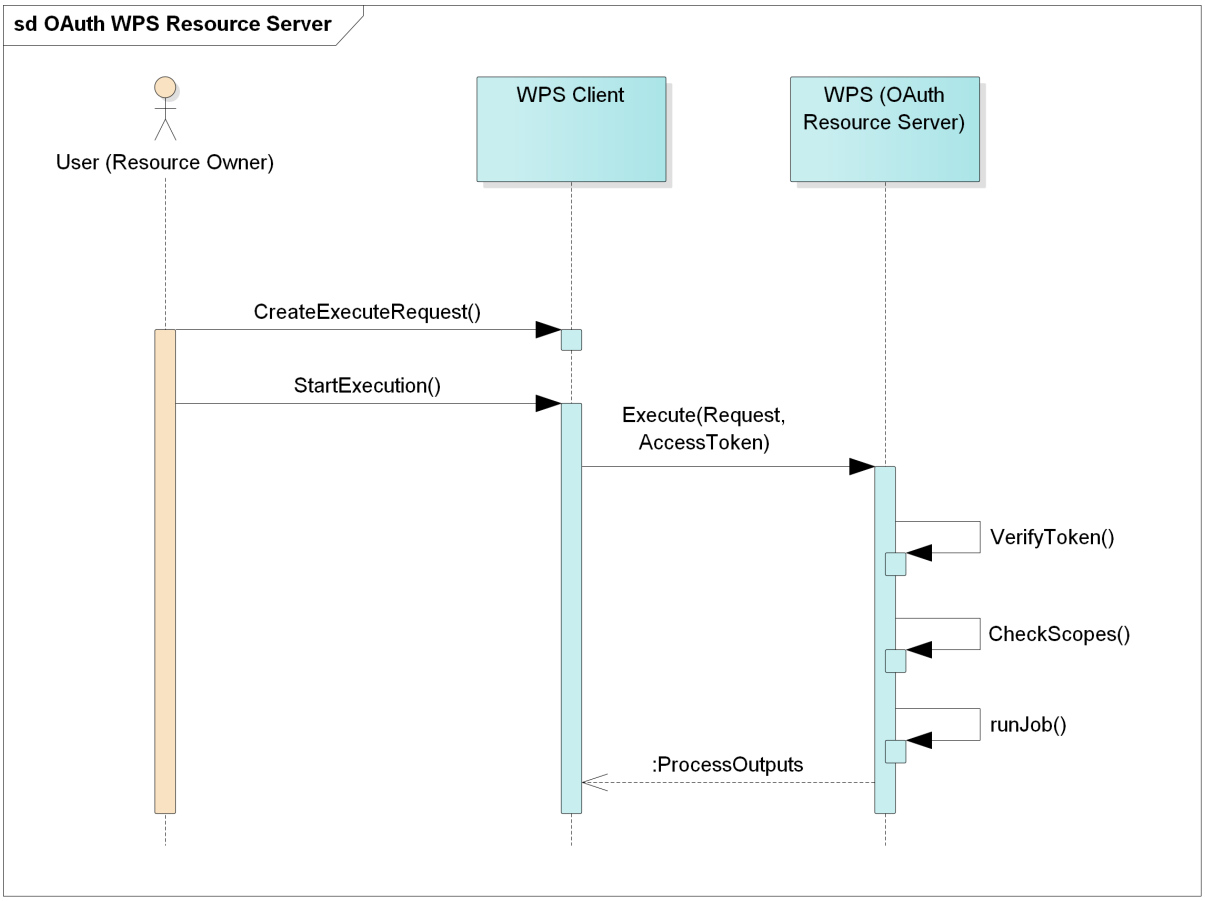


*Figure 18. Sequence diagram illustrating the WPS as OAuth resource server.*

In this case, the WPS needs to check, whether an access token arrives with an Execute request (provided in the HTTP header). Afterwards it will verify the access token and check whether the scopes allows it to execute a process.

## 9.3. Implementation

**NOTE**

*Note*

More details on the implementation, screenshots of registration at Auth0, the user authorization form, etc. can be found in the OGC Testbed 13 Workflow Engineering Report OGC #17-029, Annex C.

The OAuth enabled WPS is implemented as an extension to the 52°North Web Processing Service (52N WPS) implementation. The 52N WPS is implemented in Java and supports both, the WPS 1.0 and 2.0 standard. For implementing the 52N WPS as OAuth resource server, a Security Proxy is implemented that implements the token verification as well as the check of the scopes. In addition, it injects the OAuth-related information needed by clients in the constraints element in the operations metadata. Since Auth0 is used as authorization server and Auth0 is returning JSON Web Tokens (JWT), the tokens can be verified on the client side and scopes are also provided with the tokens. The Java libraries provided by Auth0 are used to implement the token verification and extraction of scopes. The scope check is then implemented in the Policy Enforcement Point (PEP) and can be configured by a configuration file. The Security Proxy is also used for implementing the X.509 certificate authentication and authorization based on access control list in the workflow security use cases. Hence, the proxy is described in more detail in the implementation details of the workflow security (next chapter).

The implementation of the 52N WPS as OAuth client was done by an AbstractOAuthClientAlgorithm that provides an additional endpoint that is used as redirect-URL for receiving authorization grants and access tokens from the authorization server and in turn passing the access token when resolving input references pointing to OAuth Resource servers.

# Chapter 10. Security in Workflows

| | |
|---|---|
| **NOTE** | *Instructions*<br>This section contains the contributions from 52°North regarding security for workflows. |

The workflow security use cases and concepts developed in Testbed 13 are also described in the OGC Testbed-13 Workflows ER (OGC 17-029).

## 10.1. Overview

The workflow package in Testbed 13 developed a consistent, flexible, adaptable workflow that will run behind the scenes, is described in the Business Process Model and Notation (BPMN) standard, and can be shared with and executed by other users. In a nutshell, the workflow aims to automate conflation of road datasets and pre-processing steps (coordinate transformation, data quality checks). The processing steps are implemented as WPS processes and the data is passed by reference to data servers (WFS and WCS) between the different processing steps. A detailed description of the workflow scenario and the corresponding implementing components is given in the Testbed 13 - Workflow Engineering Report (OGC 17-029).

When executing geoprocessing workflows consisting of WPS processes and intermediate transactional W*S data services, it must be ensured that differing access rights and security levels can be dealt with. More specific, three major use cases are considered that need to be addressed in the TB13 workflow scenario.

**Dominating Privileges**: This use case describes the situation when a user has more privileges than the computer system being used. Access control based solely on the computer's system privileges will result in a security violation.

**Tunneling Proxies**: If a service runs behind a proxy/firewall, certain security attributes that were sent by the client might not reach the service. As an example, a mutual TLS Authentication between client and service will not work, since the proxy is creating a new connection between the service and the proxy and the client certificate will hence not reach the service.

**Identity mediation**: Many different forms of Identification and Authorization (I&A) are available. This is of importance for service-based workflows where the I&As used may differ between the workflow services. In this case, a mediation between identity and the different models used for access control needs to be implemented.

The solutions developed to address these use cases are described in the following subsection.

## 10.2. Dominating Privileges

The problem of dominating privileges occurs in the workflow scenario, if the WPS needs to access a protected resource provided by a data server, e.g. a road dataset provided in a WFS. Though the WPS may be authenticated by the data server and may have general access, operations for retrieving resources of a user may still be protected and may require authorization by the owning

user. OAuth addresses this issue, since it is made for delegating authorization from the user to applications to access the user's resources.

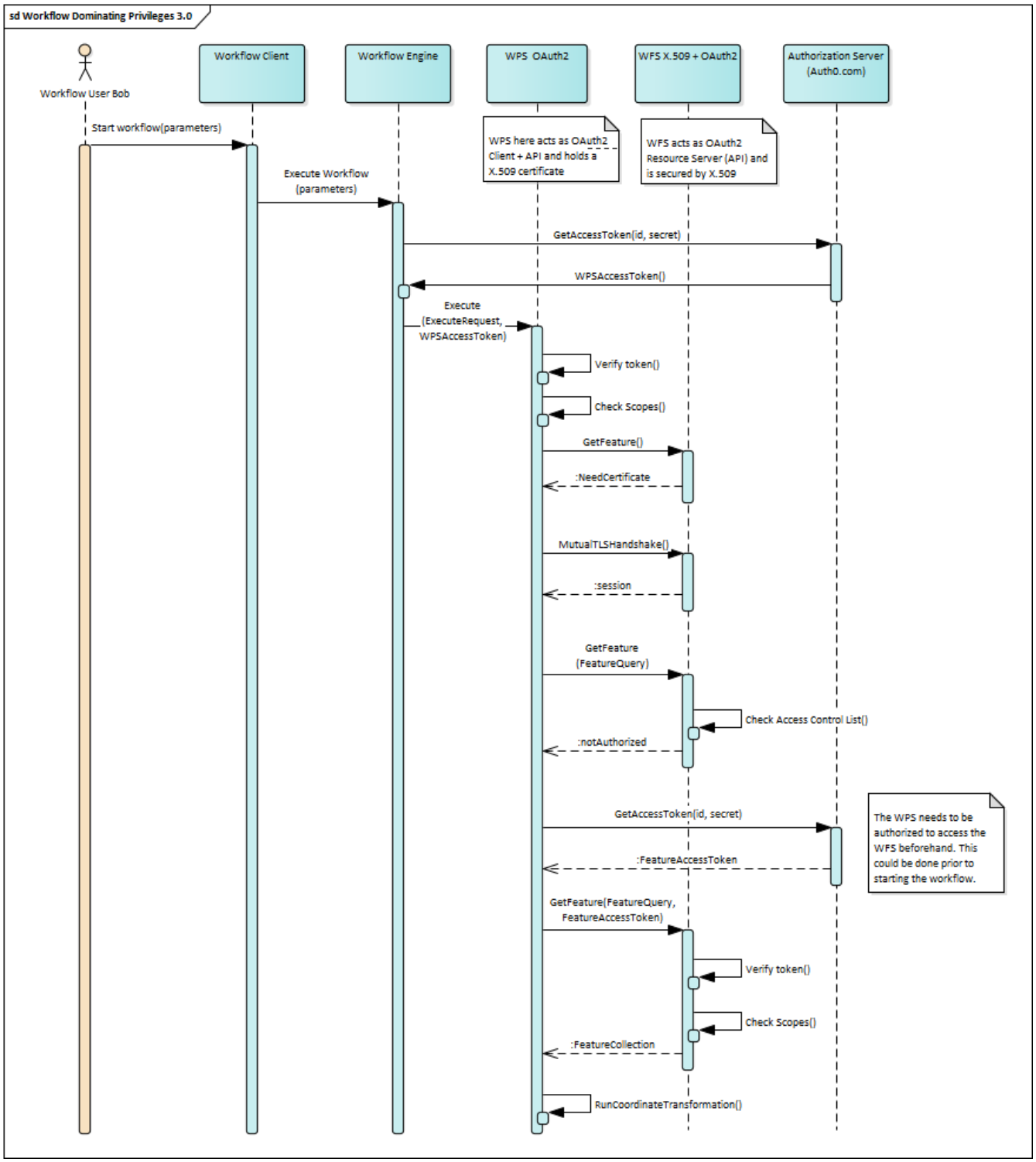


*Figure 19. Sequence diagram illustrating the concept for dominating privileges use case in Testbed 13 workflow scenario.*

In the Testbed 13 workflow, a WPS needs to retrieve a road dataset from a WFS. The road dataset is passed as reference to the WPS. The WPS hence at first queries the WFS providing the dataset. The WPS is authenticated by a X.509 certificate that allows the WPS to retrieve the Capabilities document (managed by an Access Control List), but not to retrieve the actual feature data, which is owned by the user running the whole workflow. In this case, authorization needs to be granted form the user owning resources in the WFS to the WPS. This can either be done beforehand using

the Client Credentials flow of OAuth (as shown in the diagram above), where the user has authorized the WPS beforehand to access the roads in the WFS and the WPS can directly use its client credentials to retrieve access tokens. Another option is to utilize the Authorization Code Flow to request the authorization using the authorization server and the user agent. This is described in detail in the section on the OAuth-enabled Web Processing Service above.

## 10.3. Tunneling Proxies

To address the issue that a service behind a proxy will not retrieve all client security information, consider the case when the client is authenticated by an X.509 certificate that would be not transferred through the proxy on the protocol level. Instead, the certificate needs to be provided on the application level as illustrated in the diagram below.

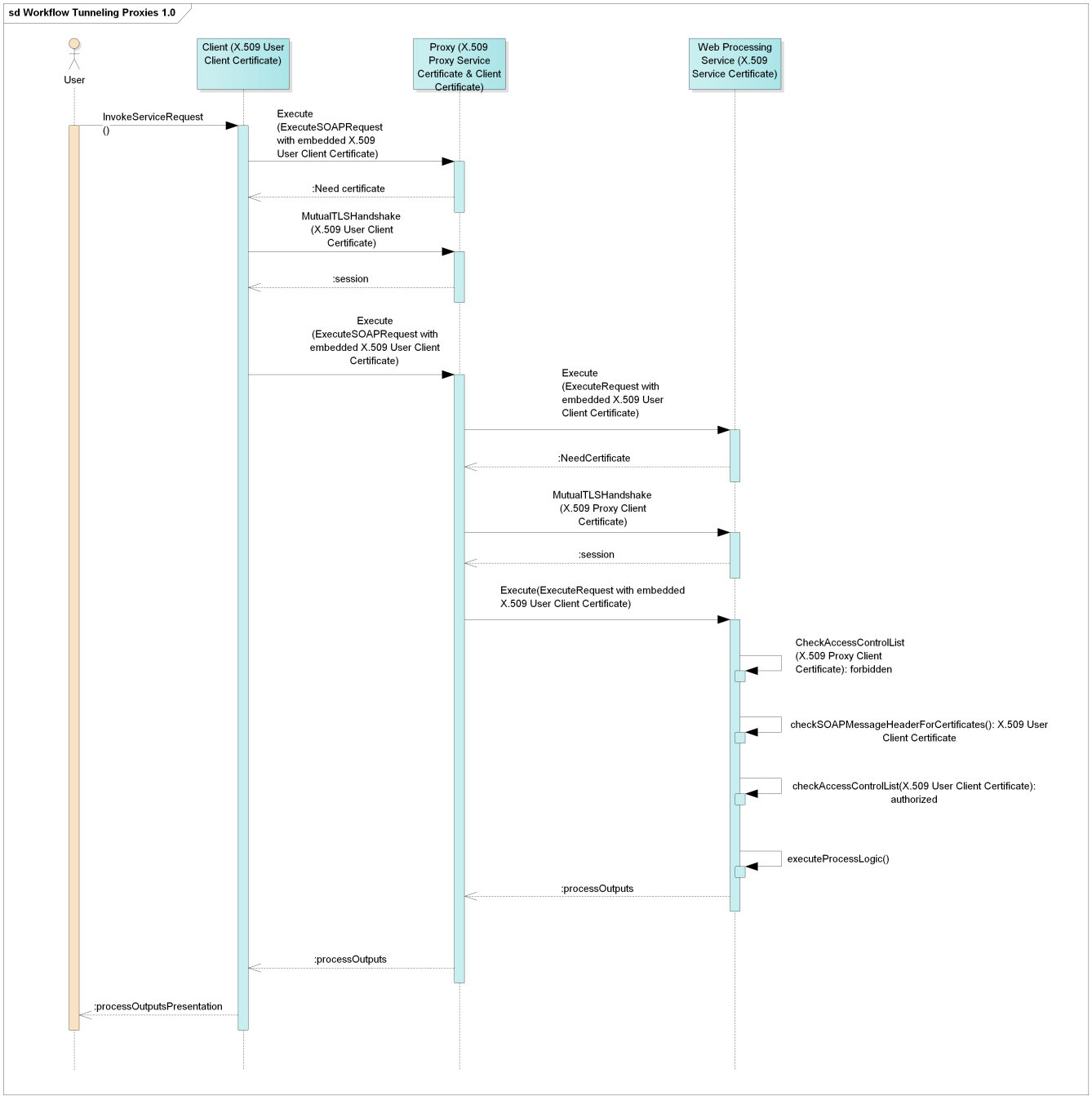


*Figure 20. Sequence diagram illustrating the concept for the tunneling proxies use case in Testbed 13 workflow scenario.*

The client holds an X.509 user client certificate and sends Execute Request to the WPS. The Execute requests reaches the Proxy at first. The Proxy requires the client to provide its certificate. The Proxy holds two certificates, a X.509 server certificate and a X.509 client certificate that is used to communicate with services behind proxy. After the secured session between client and proxy is established, the client can send an Execute request again. The execute request is now forwarded to the WPS. Again, the WPS asks for a client certificate. On the protocol level, the secured session between Proxy and WPS is established with the Proxy Client Certificate and WPS Service Certificate. But for authenticating the actual client and providing access checking the ACL, the WPS needs the user client certificate.

The solution to the problem is to transfer the client certificate on the application level. A standard that can be utilized is the Web Services Security X.509 Certificate Token Profile that describes how to embed X.509 certificates in SOAPMessageHeaders. Hence, if the proxy client certificate is not allowed to run the actual process execution, the WPS can check, whether a certificate is contained in the SOAPMessageHeader and can then check, whether this client has permission to run the execution. In this case, the execution can be done and the process outputs are then returned.

## 10.4. Identity Mediation

In this workflow, we consider the requirement that there needs to be a mediation between X.509 certificate authentication with ACL and a road dataset that is available in a WFS running in the amazon cloud. The solution to the problem is to deploy a PEP that translates the identities and maps the attributes to roles for access control as shown in the diagram below.

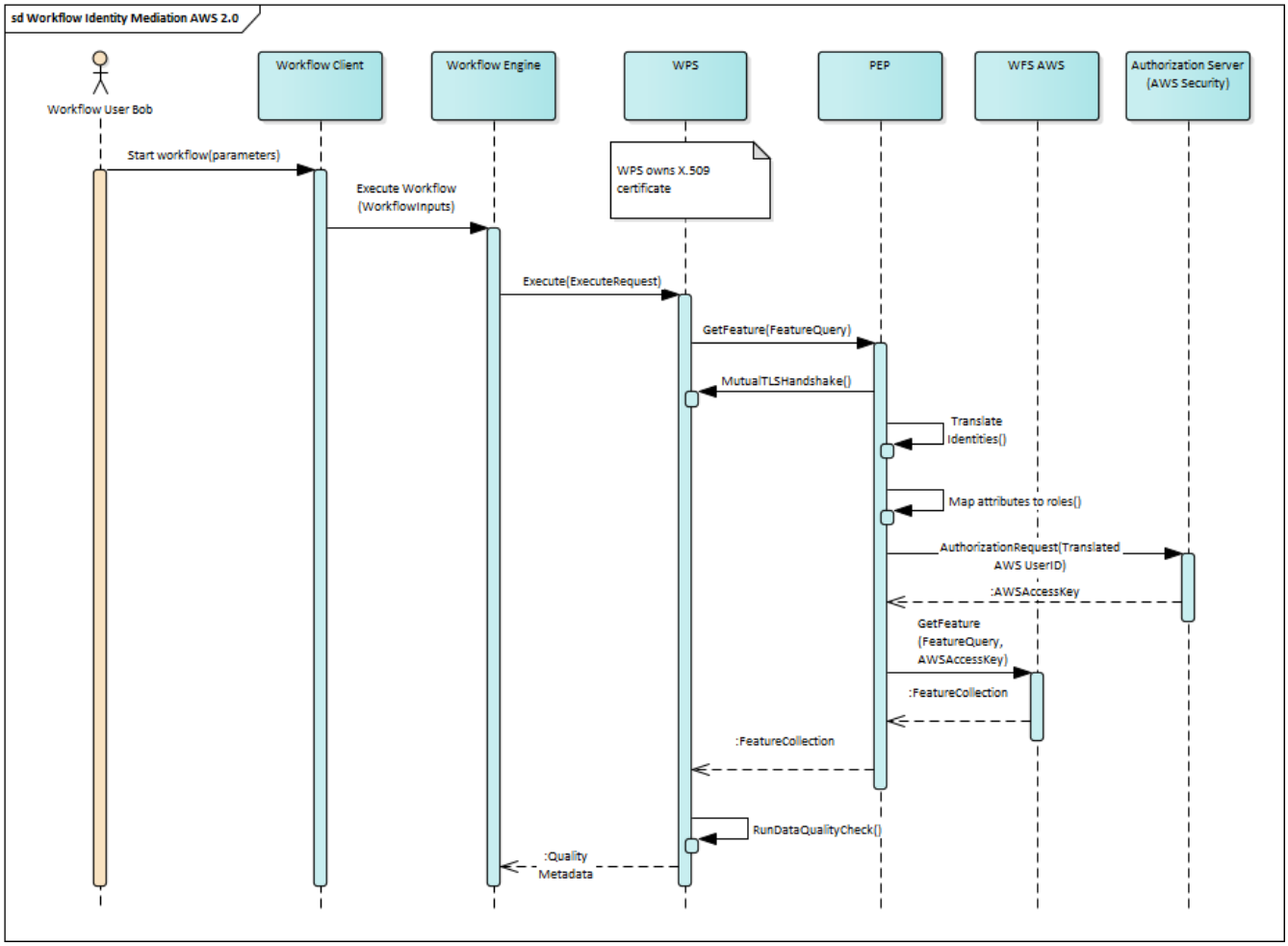


*Figure 21. Sequence diagram illustrating the concept for the identity mediation use case in Testbed 13 workflow scenario.*

The user connects to the workflow client and starts the workflow. The client triggers the workflow engine. At some point during the workflow execution, the data quality of road features is checked. A service offering the road data (WFS) is protected by AWS security. The data quality WPS holds a X.509 certificate. A PEP is used as proxy in front of the WFS. A GetFeature request is sent directly to the PEP, along with the certificate. The PEP mediates between the certificate and AWS security. Attribute based access control is mapped to role based access control. The PEP then requests AWS Authorization server for a token. Once it has received the access token, the PEP forwards GetFeature request together with the access token to the WFS and the road dataset is returned. The PEP forwards the features to the WPS. The WPS can then run the data quality check.

## 10.5. Implementation

For implementing the OAuth Resource Servers and W*S that use X.509 certificates for authentication and access control lists for authorization, 52°North implements a security proxy. The security proxy is implemented as a Java Web Service that can be deployed with OGC Web Services (52°North is providing WPS and WFS for the Testbed 13 workflow subthread).

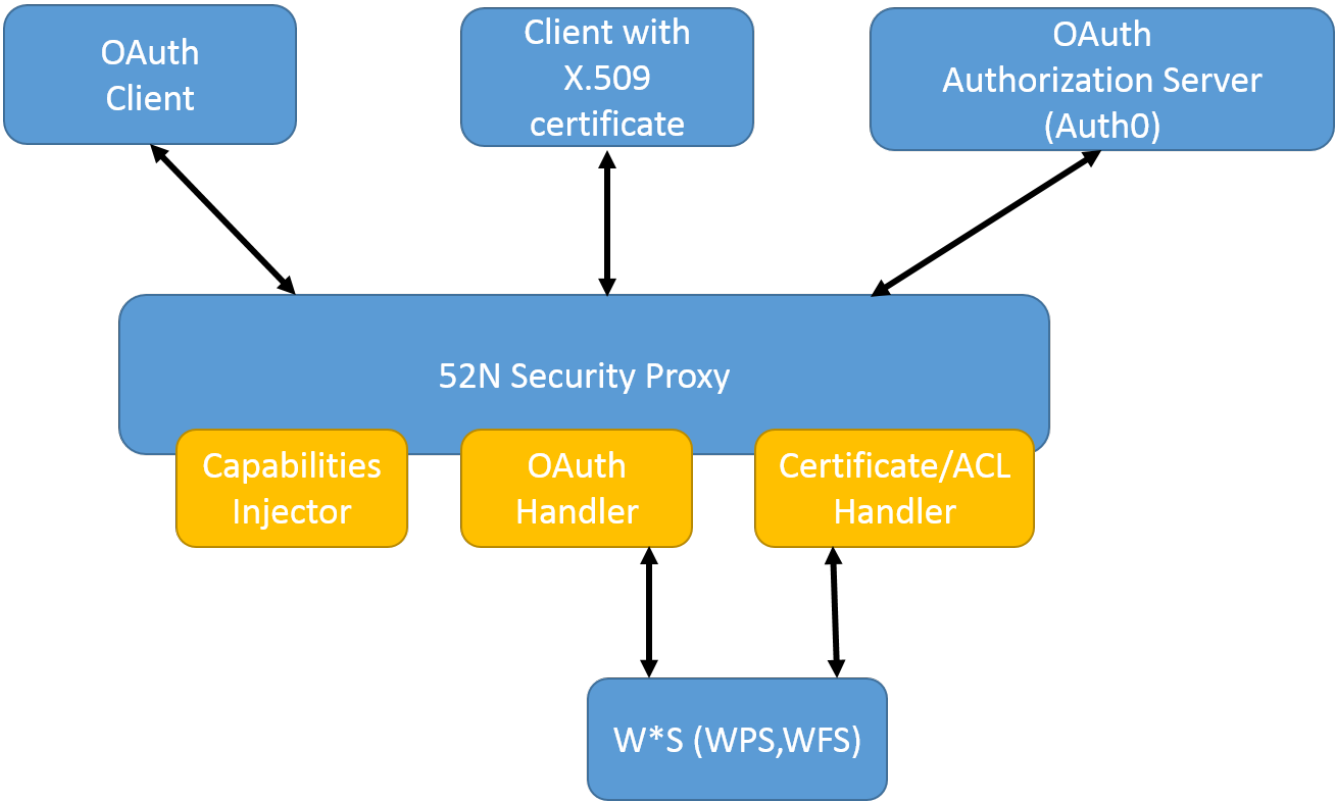


*Figure 22. Overview on the 52N security proxy implementation.*

An overview on the security proxy implementation is given in the figure above. The 52N security proxy acts as a proxy that receives requests for OGC web services, executes security checks, and, if permission is granted, forwards the requests to the OGC web services and forwards the responses back to the clients. As illustrated in the subsections above, OAuth resource servers as well as servers supporting X.509 certificate authentication and access control lists are needed to address the issues in the workflow security use cases. Thus, the security proxy implements two handlers, the OAuth handler and the Certificate/ACL handler.

The **OAuth handler** has two major tasks:

- Token Verification: It verifies the access token that is sent by the client with the service request.
- Scope Verification: The proxy also needs to verify the list of scopes associated with the access token. If the scope to perform the requested service operation is not included, the proxy denies the request and returns an HTTP 401 "Unauthorized" response with an WWW-Authenticate header indicating that a bearer token is required. If the scope is sufficient, the service request is forwarded to the actual service.

Since auth0 [2: https://auth0.com] is used as authorization server and JSON Web Tokens (JWT) are provided as access tokens, the verification is implemented in the proxy using a token verification Java library provided by auth0. This library is also used to extract the scopes and implement the scope checks. The scope itself can be configured in a configuration file.

The **Certificate/ACL handler** handles requests that include a X.509 certificate for authentication. For authorization, it implements an access control list that defines the rights of the user to access operations and/or resources in the secured service. For the implementation of the X.509 certificate authentication, we are using the Spring Security framework. The access control list can be defined in a configuration file that is read when starting the proxy.

Example of a SimplePermissions policy:

```
<SimplePermissions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.52north.org/security/simple-permission/1.0">
    <PermissionSet name="AWS1 Permission">
        <ResourceDomain value="https://s3-eu-west-1.amazonaws.com/testbed13-osm/" />
        <ActionDomain value="https://s3-eu-west-1.amazonaws.com/testbed13-osm/" />
        <SubjectDomain value="urn:n52:security:subject:role" />
        <Permission name="e.h.juerrens_GetObject">
            <Resource value="manhattan/osm-manhattan-roads.osm" />
            <Action value="GetObject" />
            <Subject value="e.h.juerrens" />
        </Permission>
    </PermissionSet>
</SimplePermissions>
```

In both cases, the **Capabilities injector** is used to add the security constraints for the service in the constraints element of the OperationsMetadata section in the Capabilities (see OGC OGC 16-048r1).

Excerpt from a WFS capabilities document describing the access constraints:

```
<ows:Operation name="GetFeature">
    <ows:DCP>
        <ows:HTTP>
            <ows:Get xlink:href="https://tb12.dev.52north.org/security-
proxy/service/wfs" />
            <ows:Post xlink:href="https://tb12.dev.52north.org/security-
proxy/service/wfs" />
        </ows:HTTP>
    </ows:DCP>
    <ows:Parameter name="resultType">
        <ows:AllowedValues>
            <ows:Value>results</ows:Value>
            <ows:Value>hits</ows:Value>
        </ows:AllowedValues>
    </ows:Parameter>
    <ows:Parameter name="outputFormat">
        <ows:AllowedValues>
            <ows:Value>text/xml; subtype=gml/3.2</ows:Value>
            <ows:Value>GML2</ows:Value>
            ...
            <ows:Value>KML</ows:Value>
        </ows:AllowedValues>
    </ows:Parameter>
    <ows:Constraint name="PagingIsTransactionSafe">
        <ows:NoValues />
        <ows:DefaultValue>FALSE</ows:DefaultValue>
    </ows:Constraint>
    <ows:Constraint name="CountDefault">
```

```
        <ows:NoValues />
        <ows:DefaultValue>1000000</ows:DefaultValue>
    </ows:Constraint>
    <ows:Constraint>
        <ows:ValuesReference
            ows:reference="https://www.tb13.secure-
dimensions.de/authnCodeList#OAUTH2_BEARER_TOKEN">
            urn:ogc:def:security:authentication:ietf:6750:Bearer
        </ows:ValuesReference>
        <ows:Metadata xlink:role="AuthorizationServer"
            xlink:href="https://bpross-52n.eu.auth0.com/authorize" />
    </ows:Constraint>
    <ows:Constraint name="urn:ogc:def:security:oauth2:scopes">
        <ows:AllowedValues>
            <ows:Value>GetFeature</ows:Value>
        </ows:AllowedValues>
        <ows:AllowedValues>
            <ows:Value>
                GetFeature/TypeName=tb13:manhattan-streets-reference
            </ows:Value>
        </ows:AllowedValues>
        <ows:AllowedValues>
            <ows:Value>GetFeature/TypeName=tb13:tnm-manhattan-streets</ows:Value>
        </ows:AllowedValues>
        <ows:AllowedValues>
            <ows:Value>GetFeature/TypeName=tb13:osm-manhattan-streets</ows:Value>
        </ows:AllowedValues>
        <ows:AllowedValues>
            <ows:Value>GetFeature/TypeName=tb13:lion-manhattan-streets</ows:Value>
        </ows:AllowedValues>
        <ows:AllowedValues>
            <ows:Value>GetFeature/TypeName=tb13:un-zataari-roads</ows:Value>
        </ows:AllowedValues>
        <ows:AllowedValues>
            <ows:Value>
                GetFeature/TypeName=tb13:tnm-manhattan-streets-wgs84
            </ows:Value>
        </ows:AllowedValues>
    </ows:Constraint>
</ows:Operation>
```

## 10.6. OAuth scopes

Scopes were defined on two levels: (1) operation and (2) feature type/process level. On the operation level, scopes were defined as Execute, DescribeProcess or GetFeature. If a client is authorized for the respective scope, it can call the operation with no limitation, e.g. call Execute on all processes or get all feature types. To allow more fine-grained access control, we defined scopes on feature type/process level. The scheme for these scopes is: Operation/AttributeName=AttributeValue, e.g. Execute/ProcessID=my.process or

DescribeFeatureType/TypeName=my.feature.

The first level of scopes works well, as the available operations of a OGC Web services instance rarely change. The second level of scopes works well for a fixed set of feature types/processes. However, adding new feature types or processes (resulting in new scopes) is quite common and this can cause a problem. The client initially requests authorization for a given set of scopes. If new scopes are added within the authorization server, the client will not be authorized to access resources connected to these scopes. It would need to make a new authorization request for each additional scope.

# Chapter 11. Technology Integration Experiments

| NOTE | *Note*<br>This section provides a summary on the TIEs done in Testbed 13 related to Security. |
|---|---|

In Testbed 13, different participants are tasked to run a Technology Integration Experiment (TIE) to demonstrate interoperability between the implementation deliverable.

## 11.1. Workflow Security

For results on TIEs in Workflow Security, please see ER OGC #17-029 "OGC Testbed 13 - Workflows ER".

## 11.2. Security Enabled Client

For results of TIEs regarding the Secured Client (AB105), please see ER OGC #17-078 "OGC Testbed 13 - Concepts of Data and Standards for Mass Migration ER".

## 11.3. QGIS Plugin

The following table provides an overview about the different TIEs and the components involved.

*Table 2. TIE components*

| Abbreviation | Component name | Deliverable |
|---|---|---|
| QGIS-PlgIn | QGIS OAuth2 Plugin | GE101 |
| 52N-RS | 52°North Resource Server with WFS | NG132 |
| Auth0.com | OAuth2 Authorization Server | linked with NG132 |
| DC-AS | Dutch Kadaster Authorization Server | linked with GE101 |
| DC-RS | Dutch Kadaster Resource Server | linked with GE101 |
| SECD-AS | Secure Dimensions Authorization Server | linked with GE101 |
| SECD-RS | Secure Dimensions Resource Server | linked with GE101 |

The following table lists all the conducted TIEs.

*Table 3. TIE Pairings*

| TIE between | Security | Use Case | Success |
|---|---|---|---|
| QGIS-PlgIn / 52N-RS | OAuth2 - Authorization Code Grant | Show conflation result provided by WFS | Y |

| TIE between | Security | Use Case | Success |
|---|---|---|---|
| QGIS-PlgIn / DC-RS | OAuth2 - Resource Owner Password Credentials | Show orthophoto provided by Dutch Kadaster WMS | Y |
| QGIS-PlgIn / DC-RS | OAuth2 - Authorization Code Grant (localhost redirect_uri) | Show orthophoto provided by Dutch Kadaster WMS | Y |
| QGIS-PlgIn / SECD-RS | OAuth2 - Authorization Code Grant (localhost redirect_uri) | Show WMS layer from Geoserver | Y |
| QGIS-PlgIn / SECD-RS | OAuth2 - Authorization Code Grant (manual finish) | Show WMS layer from Geoserver | Y |
| QGIS-PlgIn / SECD-RS | OAuth2 - Resource Owner Password Credentials | Show WCS coverage from Geoserver | Y |

## 11.4. Summary

The TIEs for the QGIS OAuth2 plugin are all successful.

The TIE results for Workflow can be found in OGC #17-029 "OGC Testbed 13 - Workflows ER".

# Chapter 12. Conclusions and Future Work

| | |
|---|---|
| **NOTE** | *Note*<br>This section provides a conclusion on the topics addressed in this ER. It also introduces next step options for future work in OGC Testbed initiatives and the standardization work. |

## 12.1. Conclusions

This ER shows that OGC Web Services can be protected with main stream IT security mechanisms to be consumed with a QGIS plugin but also participate in a secured workflow. For more conclusions on the workflow part, please take a look at that Engineering Report (OGC #17-029).

The work item GE101 was completed successfully. It did not only conclude in a "straight-forward" implementation but raised ground truth questions (and triggered answers) on the legitimate, appropriate and secure use of OAuth2 with open source desktop applications; aka the QGIS plugin for OAuth2. One of the fundamental recommendations of this ER is that the OAuth2 plugin should be registered with Authorization Servers via digitally signed software statements. One obvious reason is to ease the registration part for the user. The standard compliant software statements would be crafted by an admin of the Resource Server - perhaps in conjunction with the Authorization Server - and then published on the website. The publication of different software statements could reflect different access scopes but also require certain OAuth2 flows that are sanitized.

Regarding the applicability of the different OAuth2 grant types with the QGIS plugin, it seems to be best practice not to use the Implicit grant as the plugin cannot request new access tokens, because the flow does not issue refresh tokens. Also, the use of Authorization Code Grant with manual finish was implemented which can be recommended for computing environments where the plugin cannot create a localhost socket. This might be very relevant for restricted computing environments.

OAuth enabled WPS: Using the OAuth2 Client Credentials Flow worked well with the WPS interface standard. Enabling the WPS to access OAuth2 protected resources worked directly and the standard doesn’t need to be changed. Just some work "under the hood" was needed to resolve OAuth2 protected referenced data, e.g. request tokens from an Authorization Server and adding HTTP headers to data-requests. However, access needed to be granted beforehand. The more dynamic Authorization Code Flow also worked with WPS, but some extensions were needed. The redirect-URL (see Authorization Code Flow vs. Client Credentials Flow for details) needs to be made available to the WPS-client. This could be done by extending the GetStatus document or by returning HTTP status code 302 (Found) and HTTP header *Location*. Both approaches would require changes to the WPS interface standard and their feasibility needs to be discussed in the SWG. Another future work item would be the definition of OAuth2 scopes and the granularity of access control that they should be used for.

## 12.2. Implications to OGC Standardization Efforts

Results from the documented work regarding security in OGC Testbed 13 indicates that OGC and main stream IT security approaches work well together. The topics addressed will further stimulate sustainable discussions and uptake in the Security DWG.

Standardization activities in the OWS Common - Security SWG benefit from the results as it is now clearer which annotations to do for an OAuth2 protected service and in how far that would further improve (ease) the plugin registration activity. Furthermore, the results from Testbed 13 indicate that the OWS Common - Security SWG approach on defining a backwards compatible approach is good practice. However, for future work it is very important that security gets built into the standardization effort as early as possible. The new upcoming standardization work with WFS/FES as API provides opportunities that should be followed.

## 12.3. Future Work

Based upon the results from Testbed 13 but also current standardization activities in the OGC, this ER defines the following different future work topics.

### 12.3.1. QGIS plugin to parse security annotations in Capabilities

Based on the result to secure OWS Web Services with modern IT security standards such as OAuth2 and SAML2, a direct follow up could be to implement parsing and processing of the proposed security annotations for Capabilities documents as identified by the OGC OWS Common - Security SWG. Based on the OAuth2 and SAML2 (but potentially other authentication methods) supported by the QGIS application, a challenge could be to implement for a WMS / WFS provider the ability to parse the security annotations in the capabilities response and execute the authentication plugin to trigger authentication configuration. Based on the parsing and processing, it should be possible to provide already certain information based on the security annotations. For example, the OWS Common - Security SWG proposes to use a SAML2 constraint that contains the URL of the associated SAML2 Federation Metadata. With this input, the SAML2 plugin configuration could already present the user a list of available IdPs to choose from. It would not be necessary to ask the user to provide the Federation Metadata URL as manual input as illustrated in figure "SAML2 Plugin configuration example".

### 12.3.2. OGC Web Services Security Standard

| | |
|---|---|
| **NOTE** | *Note*<br>OGC Web Services Security Standard is currently available as SWG working draft. |

The OGC Web Services Security standard describes tests for validation of the Conformance Classes and the associated Requirements Classes. In essence, the standard requires a test harness to validate a client and a service implementation. Future work in an OGC Testbed could be to implement such a test harness; one for testing client compliance and one for testing service compliance. Before implementing such a test harness from scratch, it would be ideal to evaluate the existing OGC CITE harness. But potentially this test harness is not suitable for testing security requirements, as the objective is not to test conformance with other OGC Implementation

Specifications but compliance with main stream IT security standards. For example, the security test harness for the service would check whether an OAuth2 protected service would return the HTTP status 401 and a HTTP header WWW-Authenticate compliant to RFC 6750. For testing clients, it would mainly be important to verify that missing security annotations (and thereby lack of support for a required functionality in security) in the capabilities will cause a client failure. For example, if the client is a Web-App implemented in JavaScript, then the service must support Cross-Origin Resource Sharing (CORS) to guarantee that application request issued by the JavaScript library can pass through the Web Browser.

### 12.3.3. OGC WFS/FES SWG

| | |
|---|---|
| **NOTE** | *Note*<br>The OGC WFS/FES SWG is work in progress. |

The OGC WFS/FES SWG is about to apply main stream IT concepts for describing APIs to Web Feature Service. The main idea behind the approach is to base the interface description on an open standard: OpenAPI 3.0. The use of OpenAPI allows also to specify security requirements by leveraging so called security schemes. OpenAPI 3.0 has a set of pre-defined security schemes that allow a compliant description for Basic Authentication, Bearer Authentication, OAuth2 etc. It could be a future work item in an OGC Testbed to verify if additional security schemes would need to be defined. Also, a prototype implementation of the OpenAPI based description for a WFS **including** security schemes could be interesting. The SWG would benefit from these activities to validate their concepts and integrate security right from the start.

### 12.3.4. Federated Identity Management and Secure Access to OGC Web Services

In Testbed 13, modern security standards like OAuth2 and SAML2 were used to protect OGC Web Services. It could be a challenging work item to combine SAML2 authentication and OAuth2 Access Tokens to build a Spatial Data Infrastructure with protected services to support federated identity management.

### 12.3.5. Integrity on XML Encoded Documents

Another future topic, disconnected from the current testbed activities but linked with the security considerations currently being produced by the OWS Common - Security SWG, could be to evaluate / study the options to introduce a W3C Digital Signature on OGC encoding standards. This would provide the ability to protect a Capabilities instance document, a FeatureCollection, a WMS Context document independent from the underlying communication protocol. The XML instance documents could be sent via e-mail or put on shred drives as digital signature on the documents would protected them for undetected tampering. It would further be possible to liaise an issuer with those offline instance documents.

From a technical perspective, it would be quite simple to support a digital signature on the Encoding documents by introducing an optional element called Signature in the W3C DSID namespace to the appropriate XML schemas maintained by OGC. Because the Signature element is optional, it must not be used which guarantees backwards compatibility.

Even though the technical solution is quite simple and straight- forward, activities in future testbeds should evaluate which schemas to change and look into the exact implications to existing implementations, before recommending this "simple" change.

# Appendix A: Technology Integration Tests

This section describes the TIEs in Testbed 13 for the QGIS Authentication plugin supporting the OAuth2 access delegation (authorization) framework.

## 1.1. QGIS OAuth2 Plugin TIE

For the QGIS 2.18.4 OAuth2 Plugin, the following TIEs exist:

*Table 4. Secure Dimensions OAuth2 Resource Server*

| | |
|---|---|
| **RS URL** | https://rs.tb13.secure-dimensions.de/geoserver/tiger/wms?version=1.1.0 |
| **AS** | https://as.tb13.secure-dimensions.de |
| **Test purpose:** | Confirm that access is possible with access tokens from the AS |
| **Test method:** | Use the QGIS OAuth2 plugin with configuration for TB13 Secure Dimensions Authorization Server<br>• Use the QGIS OAuth2 plugin to configure a setup for the SECD Authorization Server with Authorization Code Grant<br>• Configure the WMS endpoint in QGIS<br>• Add the Authentication via the SECD Authorization Server to the WMS configuration<br>• Request Capabilities from the SECD RS<br>• Select any layer (e.g. Manhattan Roads) and request the map<br>• Verify if a map is received and displayed |
| **Test type:** | OAuth2 Plugin Authentication with WMS and Authorization Code Grant |

The TIE as described in table 1.1 is between the OAuth2 QGIS plugin and a WMS secured via an OAuth2 RS where the associated Authorization Server supports the Authorization Code Grant.

*Table 5. Secure Dimensions OAuth2 Resource Server*

| | |
|---|---|
| **RS URL** | https://rs.tb13.secure-dimensions.de/geoserver/wcs |
| **AS** | https://as.tb13.secure-dimensions.de |
| **Test purpose:** | Confirm that access is possible with access tokens from the AS |

| | |
|---|---|
| **Test method:** | Use the QGIS OAuth2 plugin with configuration for TB13 Secure Dimensions Authorization Server<br>• Use the QGIS OAuth2 plugin to configure a setup for the SECD Authorization Server<br>• Configure the WCS endpoint in QGIS<br>• Add the Authentication via the SECD Authorization Server to the WCS configuration<br>• Request Capabilities from the SECD RS<br>• Select any layer (e.g. nurc:mosaic) and request the map<br>• Verify if a map is received and displayed |
| **Test type:** | OAuth2 Plugin Authentication with WCS and ROPC |

The TIE as described in table 1.2 is between the OAuth2 QGIS plugin and a WCS secured via an OAuth2 RS where the associated Authorization Server supports the Resource Owner Password Credentials.

*Table 6. Dutch Kadaster OAuth2 Resource Server*

| | |
|---|---|
| **RS URL** | https://test.secure.geodata2.nationaalgeoregister.nl/lv-beeldmateriaal-poc/2016/wms?version=1.1.1 |
| **AS** | https://authorization.test.kadaster.nl |
| **Test purpose:** | Confirm that access is possible with access tokens from the AS |
| **Test method:** | Use the QGiS OAuth2 plugin with configuration for TB13 Dutch Kadaster Authorization Server<br>• Use the QGIS OAuth2 plugin to configure a setup for the Dutch Kadaster Authorization Server using ROPC<br>• Configure the WMS endpoint in QGIS<br>• Add the Authentication via the Dutch Kasdaster Authorization Server to the WMS configuration<br>• Request Capabilities from the Dutch Kadaster RS<br>• Select any layer (e.g. LV_Beeldmateriaal_2016_Ortho10) and request the map<br>• Verify if a map is received and displayed |
| **Test type:** | OAuth2 Plugin Authentication with WMS and ROPC |

The TIE as described in table 1.3 is between the OAuth2 QGIS plugin and a WMS secured via an OAuth2 RS where the associated Authorization Server supports the Resource Owner Password Credentials.

*Table 7. 52North OAuth2 Resource Server*

| | |
|---|---|
| **RS URL** | http://tb12.dev.52north.org/security-proxy/service/wfs |
| **AS** | https://bpross-52n.eu.auth0.com |
| **Test purpose:** | Confirm that access is possible with access tokens from the AS |
| **Test method:** | Use the QGiS OAuth2 plugin with configuration for TB13 Auth0.com Authorization Server<br>• Use the QGIS OAuth2 plugin to configure a setup for the Auth0.com Authorization Server using Authorization Code Grant<br>• Configure the WFS endpoint in QGIS<br>• Add the Authentication via the Auth0.com Authorization Server to the WMS configuration<br>• Request Capabilities from the 52North RS<br>• Select the layer tb13:osm-manhattan-streets and request the features<br>• Verify if a feature collection is received and displayed |
| **Test type:** | OAuth2 Plugin Authentication with WFS and Authorization Code Grant |

The TIE as described in table 1.4 is between the OAuth2 QGIS plugin and a WFS secured via an OAuth2 RS where the associated Authorization Server supports the Authorization Code Grant.

For all tests, a Windows 10, 32bit executable for the QGIS QGIS 2.18.4 and the OAuth2 plugin are used. The built "Munich" is provided by Secure Dimensions specifically for the OGC Testbed 13 but based on the Github master for version 2.18.4. This ensures compatibility with the plugin as that is installed as a DLL. The QGIS splash screen reflects that:


*Figure 23. QGIS 2.18.4 build for TB13*

### 1.1.1. TIE with SECD RS and AS using Authorization Code Grant

Once the QGIS plugin is registered with the AS, the required parameters must be provided into the

plugin GUI.

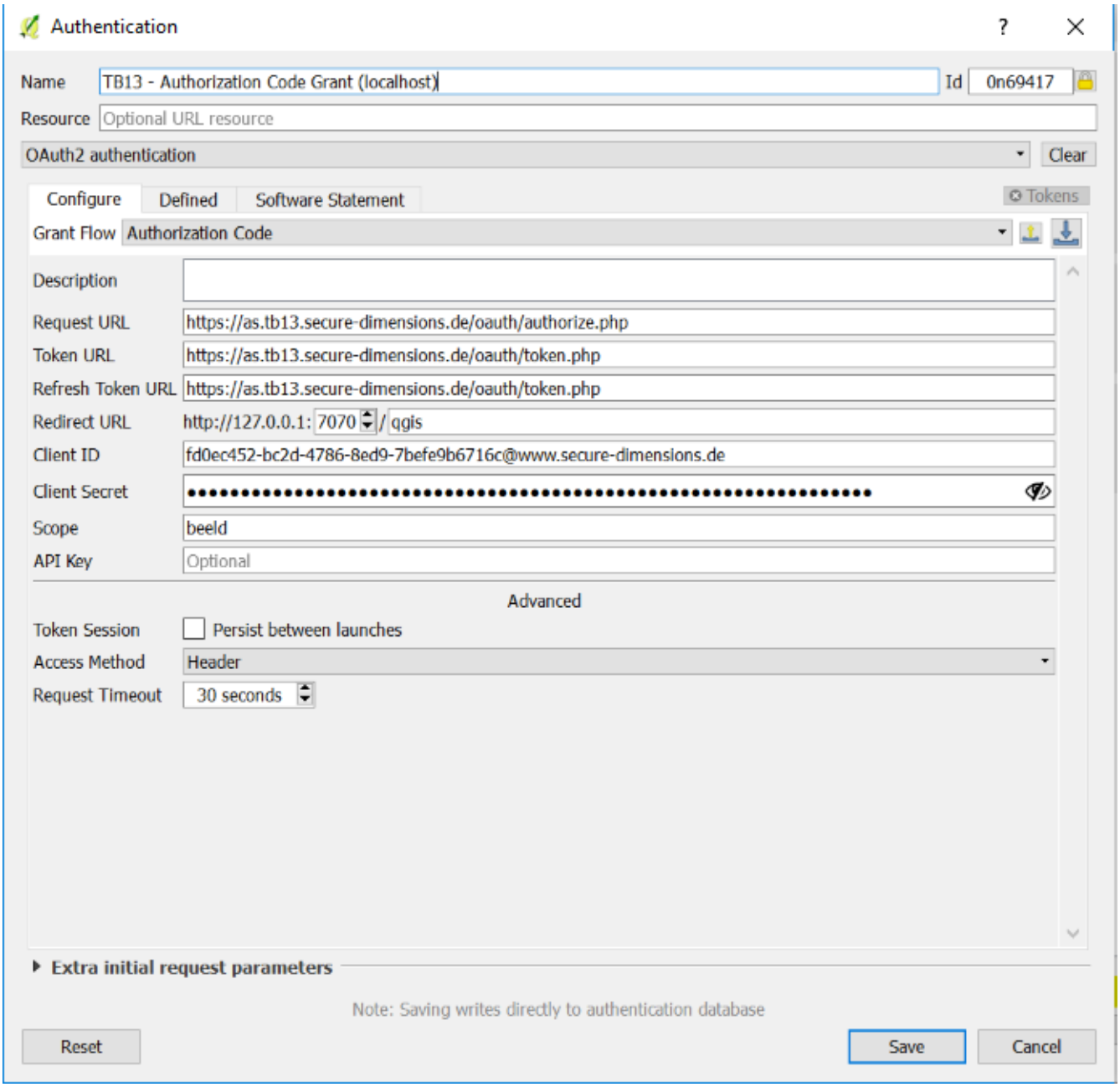


*Figure 24. TB13 Secure Dimensions AS - Authorization Code Grant*

Figure 24 illustrates the configuration of the plugin regarding the TIE as defined in table 1.1. The configuration is saved under the title provided in the name field: "TB13 - Authorization Code Grant (localhost)". In addition to the Authorization Server specific endpoints, the configuration indicates that the plugin leverages the Authorization Code Grant and a Resource Server sepcific scope: "beeld". Even though the Client Secret is not required for this configuration, it is provided as part of the registration process.

Figure 24 also illustrates that the access token will be attached to the HTTP request leveraging the HTTP header.

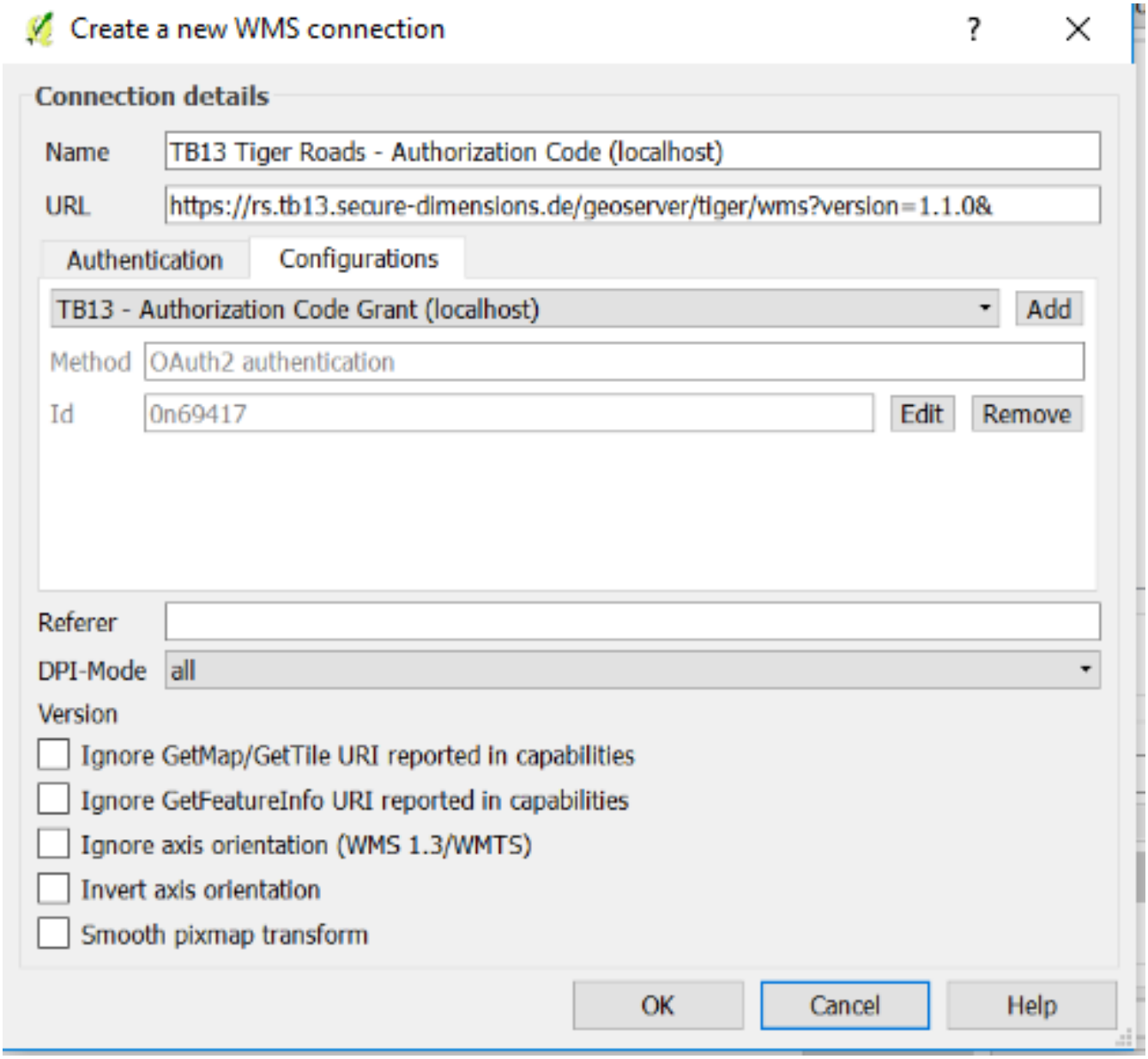


*Figure 25. QGIS configuration*

Figure 25 illustrates the use of the previous OAuth2 configuration for a particular service instance.

*Note*

The use of the Authorization Code Grant with localhost redirect URI REQUIRES that the plugin can open a port on the computer. For the example it is port 7070. The following figure illustrates the Windows Firewall warning (in German). If the user is either unable or unwilling to grant access ("Zugriff erlauben"), then this grant type cannot be used. The user must process the Authorization Code Grant with manual finish as described in a later section!

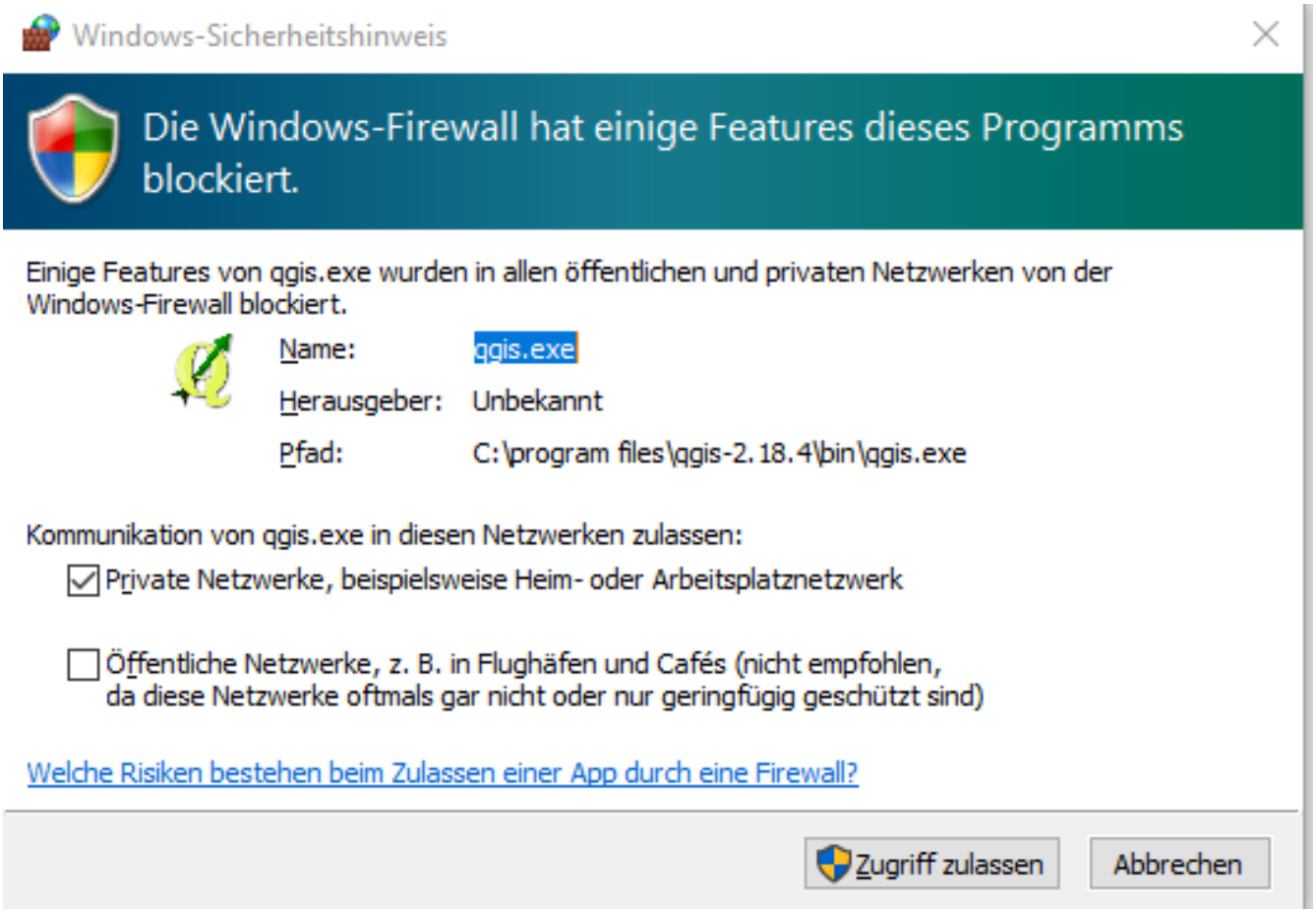


*Figure 26. Windows Firewall Confirmation Dialog*

For the Authroization Code Grant with localhost redirect_uri, the Windows user must approve the changes to the firewall. If that is not possible, the user must use the Authorization Code Grant with manual finish.

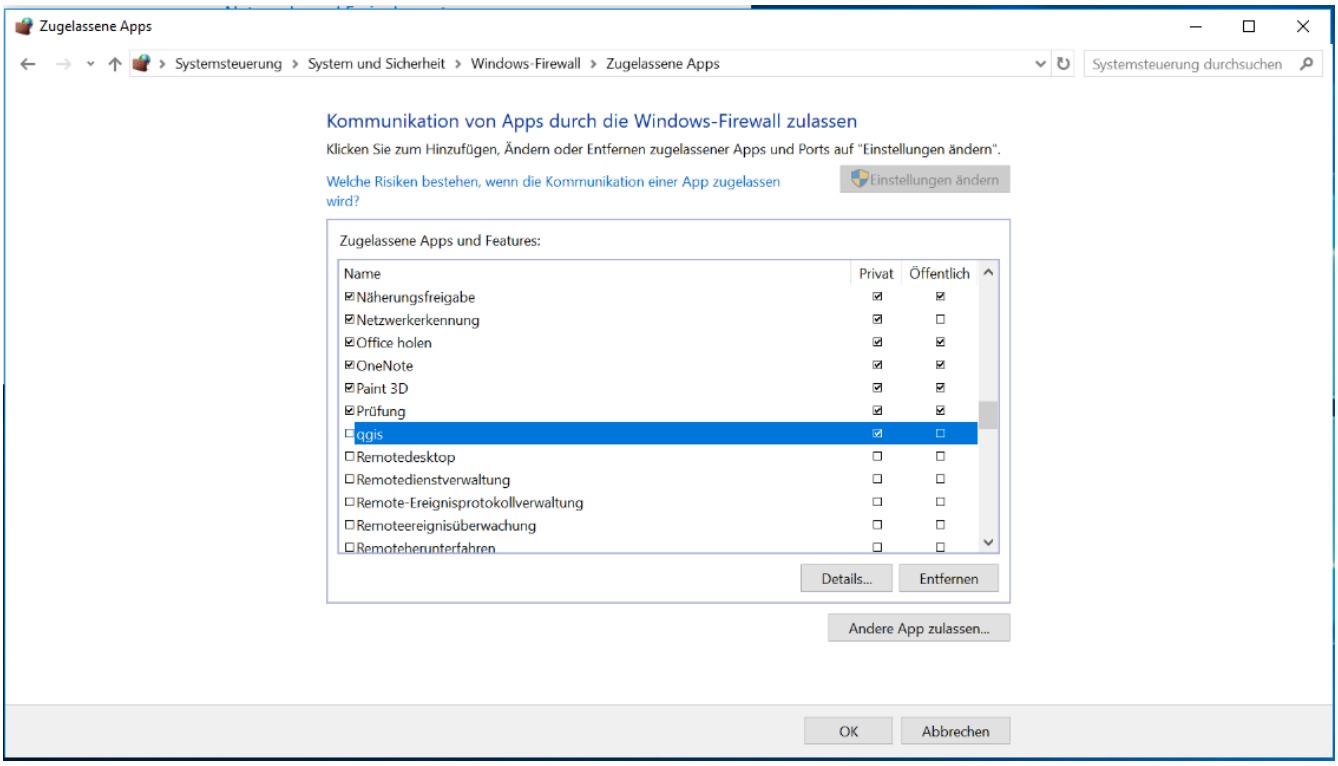


*Figure 27. Windows Firewall Change*

Assuming the user has approaved the Windows Firewall to accept the QGIS executeable to create a port on localhost, the OAuth2 flow can continue...

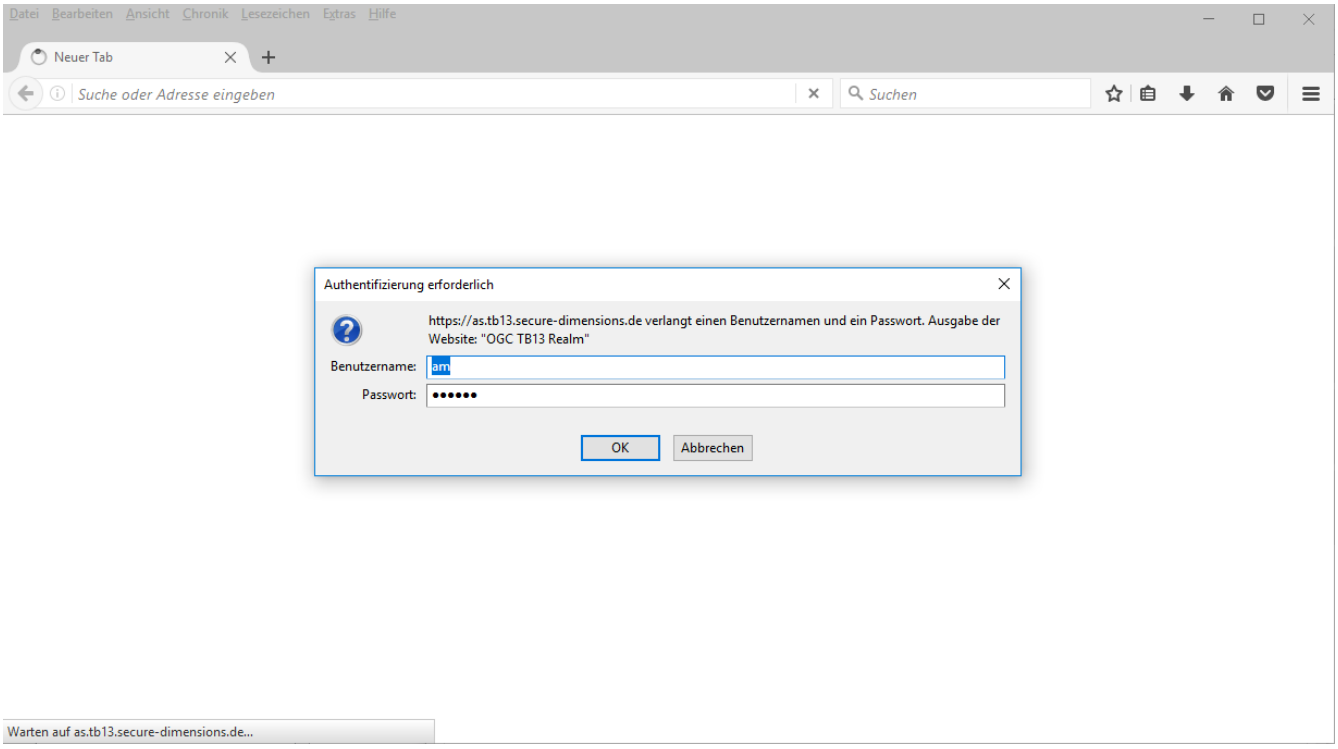


*Figure 28. TB13 Secure Dimensions AS - Login*

After the user connects to the service, the OAuth2 protocol is executed. The first step is for the user to login, as illustrated in Figure 28.

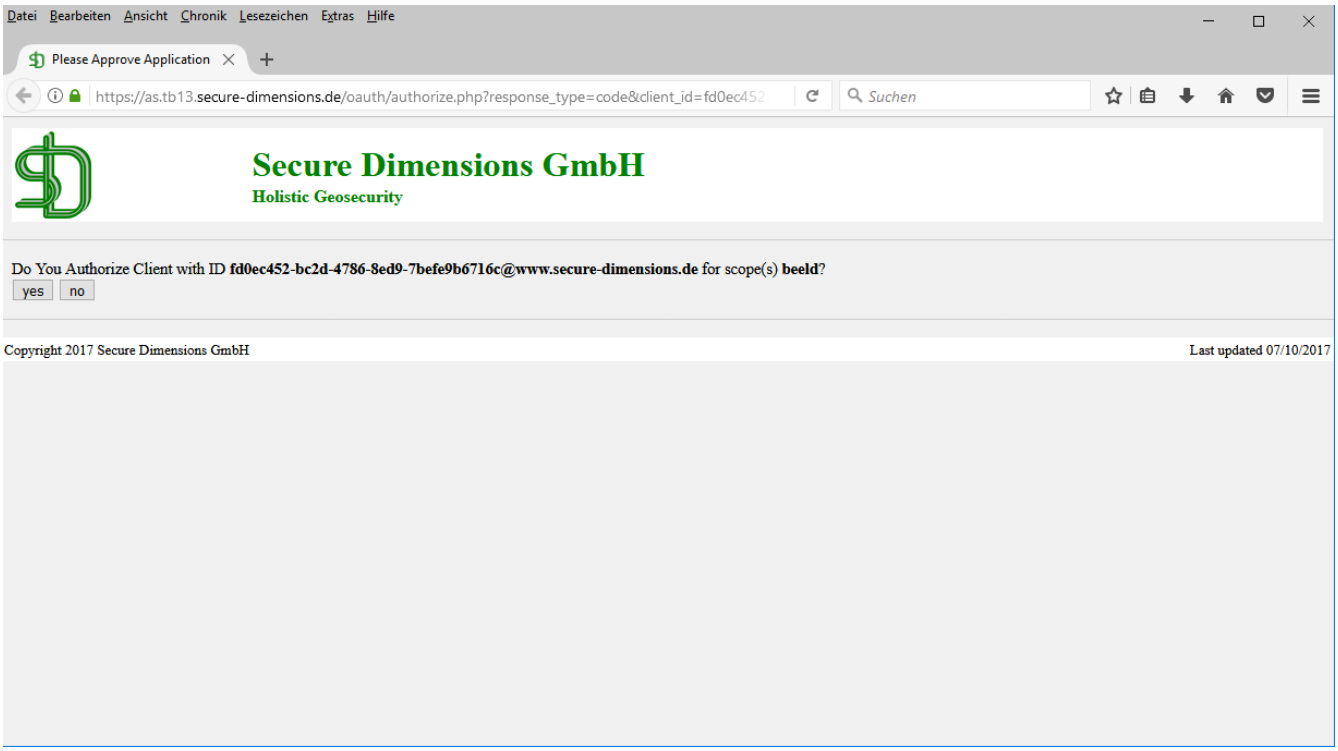


*Figure 29. TB13 Secure Dimensions AS - Approve plugin use*

Before the Authorization Server issues an authorization code (and later an access token) to the plugin, the user must approve this action. The request dialog is illustrated in figure Figure 29.

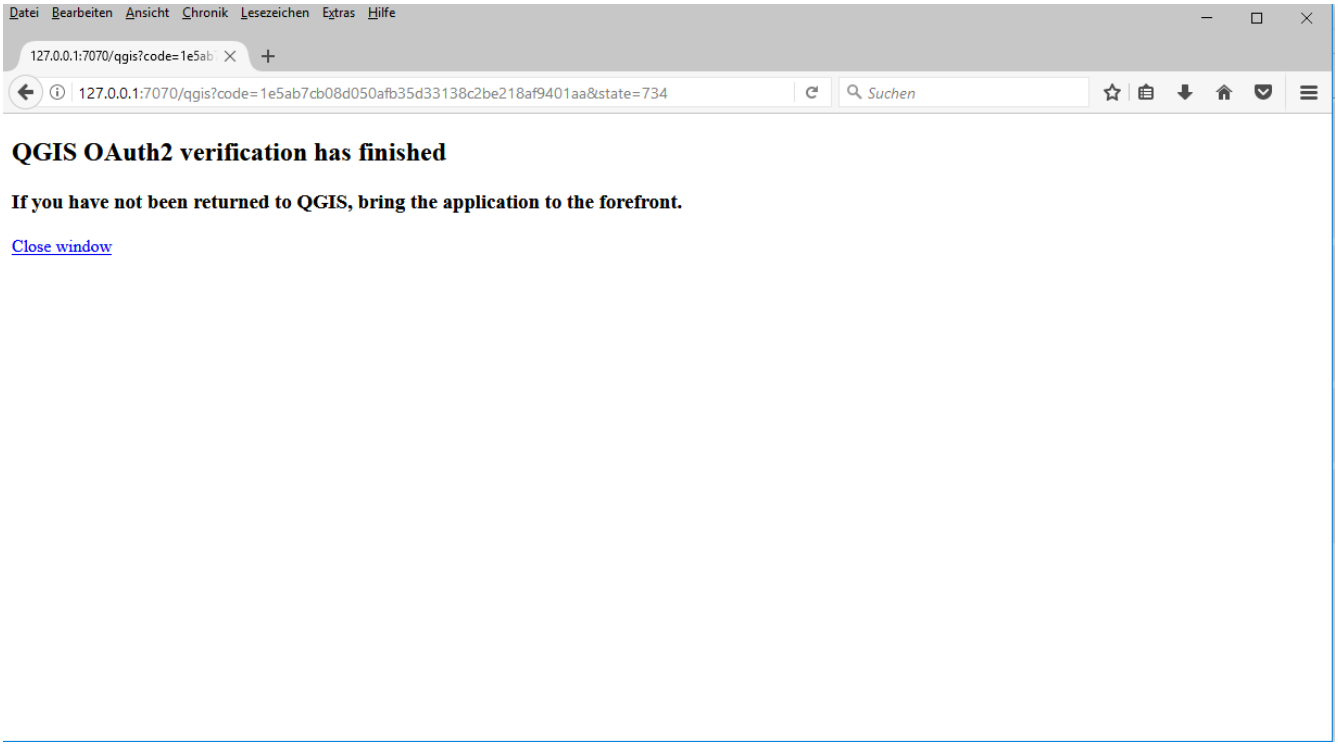


*Figure 30. Local browser - TB13 Secure Dimensions AS approval finished*

Figure 30 indicates the Authorization Code Grant has successfully completed with the Authorization Server. The user is informed to close the window (which might not work depending on the Web Browser and other configuration parameters) or focus to the QGIS application.

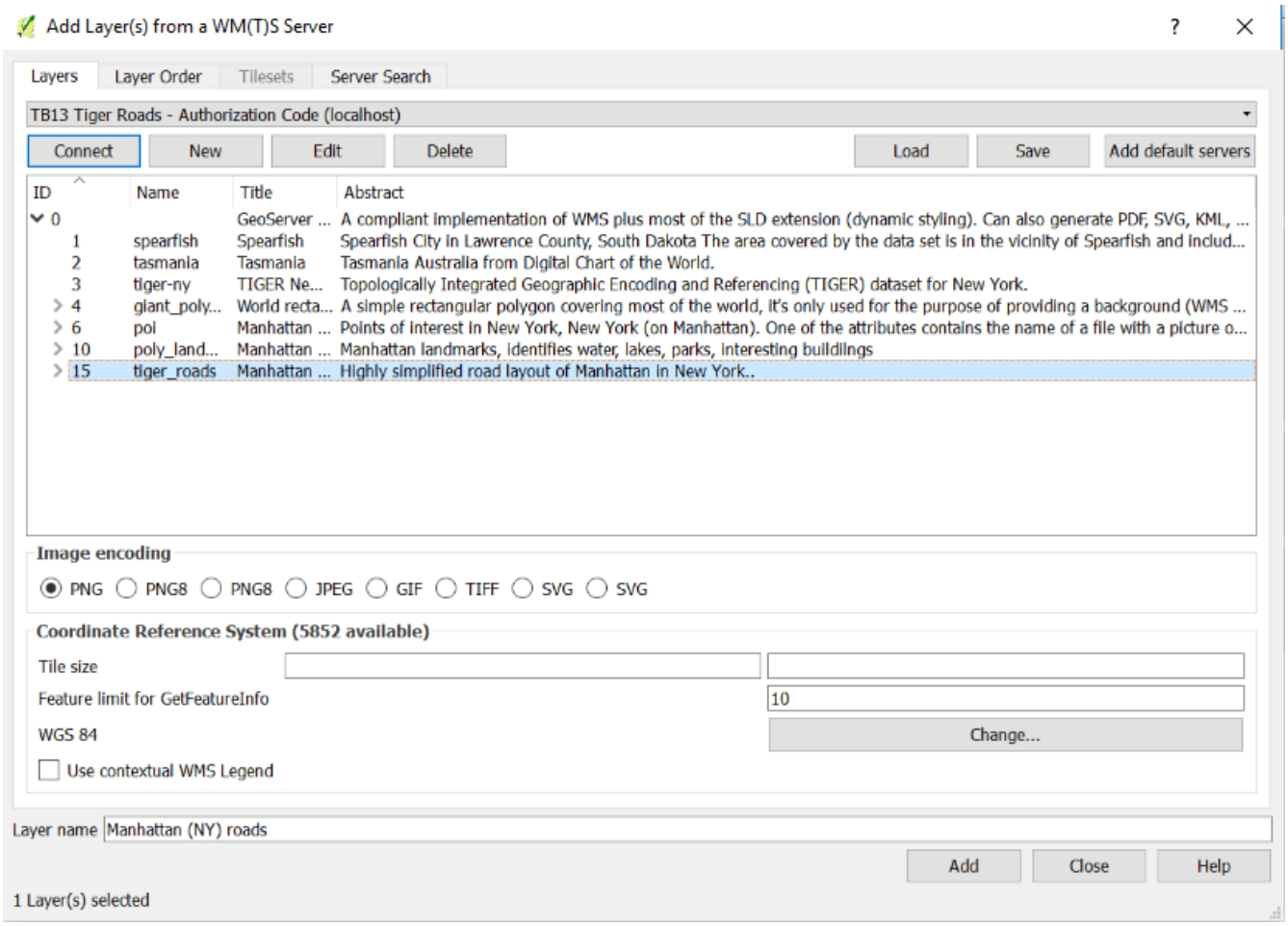


*Figure 31. TB13 Secure Dimensions RS - Capabilities Response*

Once the user returned to the QGIS application, the user must select a WMS layer. For this TIE leveraging the scope "beeld", the layer tiger_rodas can be selected as illustrated in Figure 31.

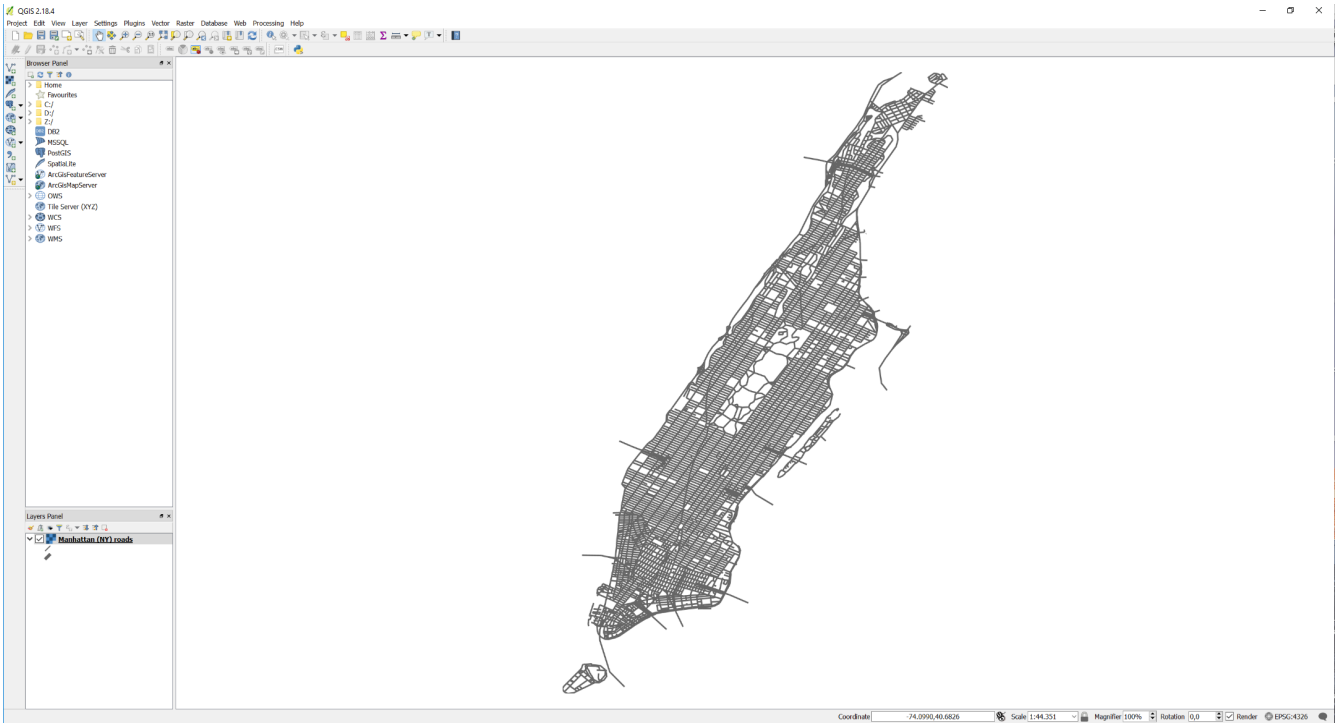


*Figure 32. QGIS displaying map from TB13 Secure Dimensions RS*

After the user has added the layer to the QGIS application, it is loaded and displayed as illustrated in Figure 32.

In case that the user executes the WMS and there is no valid access token attached to the request, the service will return an OAuth2 compliant exception as illustrated in Figure 33. GetMap request:

```
https://rs.tb13.secure-
dimensions.de/geoserver/tiger/wms?service=WMS&version=1.1.0&request=GetMap&layers=tige
r:tiger_roads&styles=&bbox=-74.02722,40.684221,-
73.907005,40.878178&width=476&height=768&srs=EPSG:4326&format=image%2Fpng
```

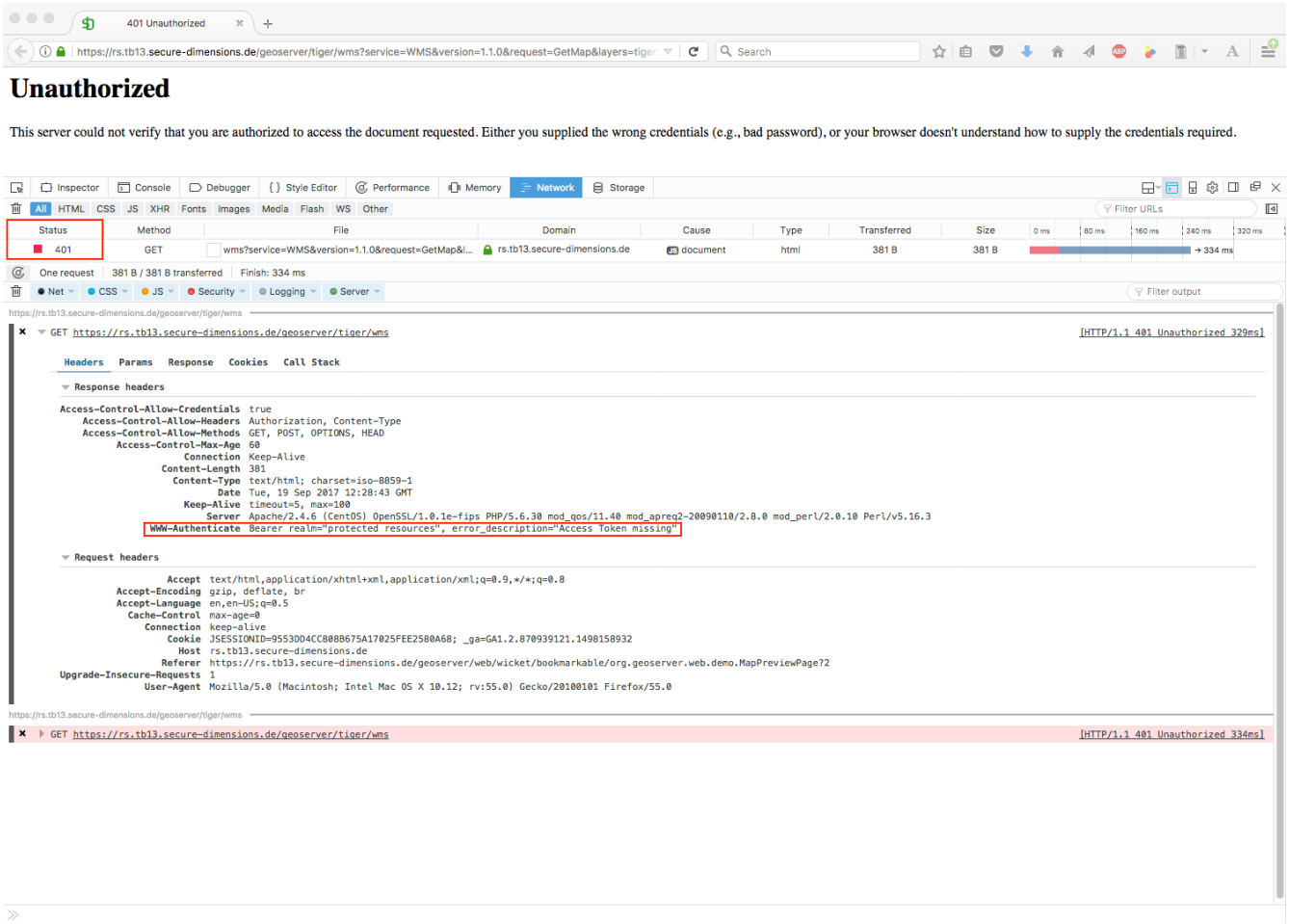


*Figure 33. Firefox displaying access token missing exception from TB13 Secure Dimensions RS*

### 1.1.2. TIE With SECD RS and AS using Authorization Code Grant with manual finish

Once the QGIS plugin is registered with the AS, the required parameters must be provided into the plugin GUI.

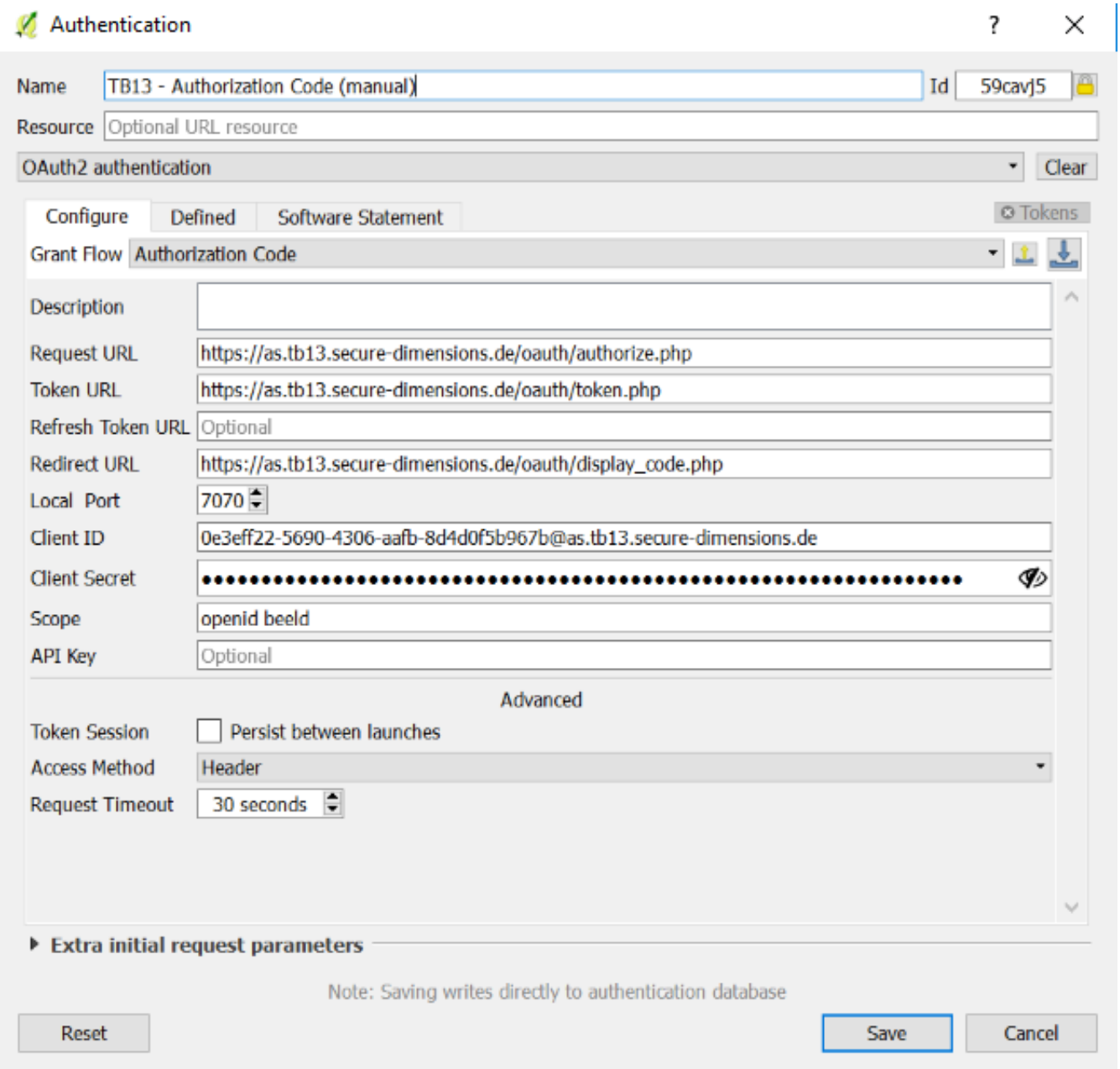


*Figure 34. TB13 Secure Dimensions AS - Authorization Code Grant Manual*

Figure 34 illustrates the configuration of the plugin regarding the TIE as defined in table 1.1. The configuration is saved under the title provided in the name field: "TB13 - Authorization Code Grant (manual)". In addition to the Authorization Server specific endpoints, the configuration indicates that the plugin leverages the Authorization Code Grant with manual finish, as the redirect URI is not localhost or 127.0.0.1.

*Note*

```
The Authorization Code Grant manual can be used when it is *not* possible that the
QGIS plugin can create a listen port on localhost. This flow does *not* require the
Windows Firewall to change! This can be important for some computing environments, for
example in the intelligence domain.
```

Figure 34 also illustrates that the access token will be attached to the HTTP request leveraging the HTTP header.

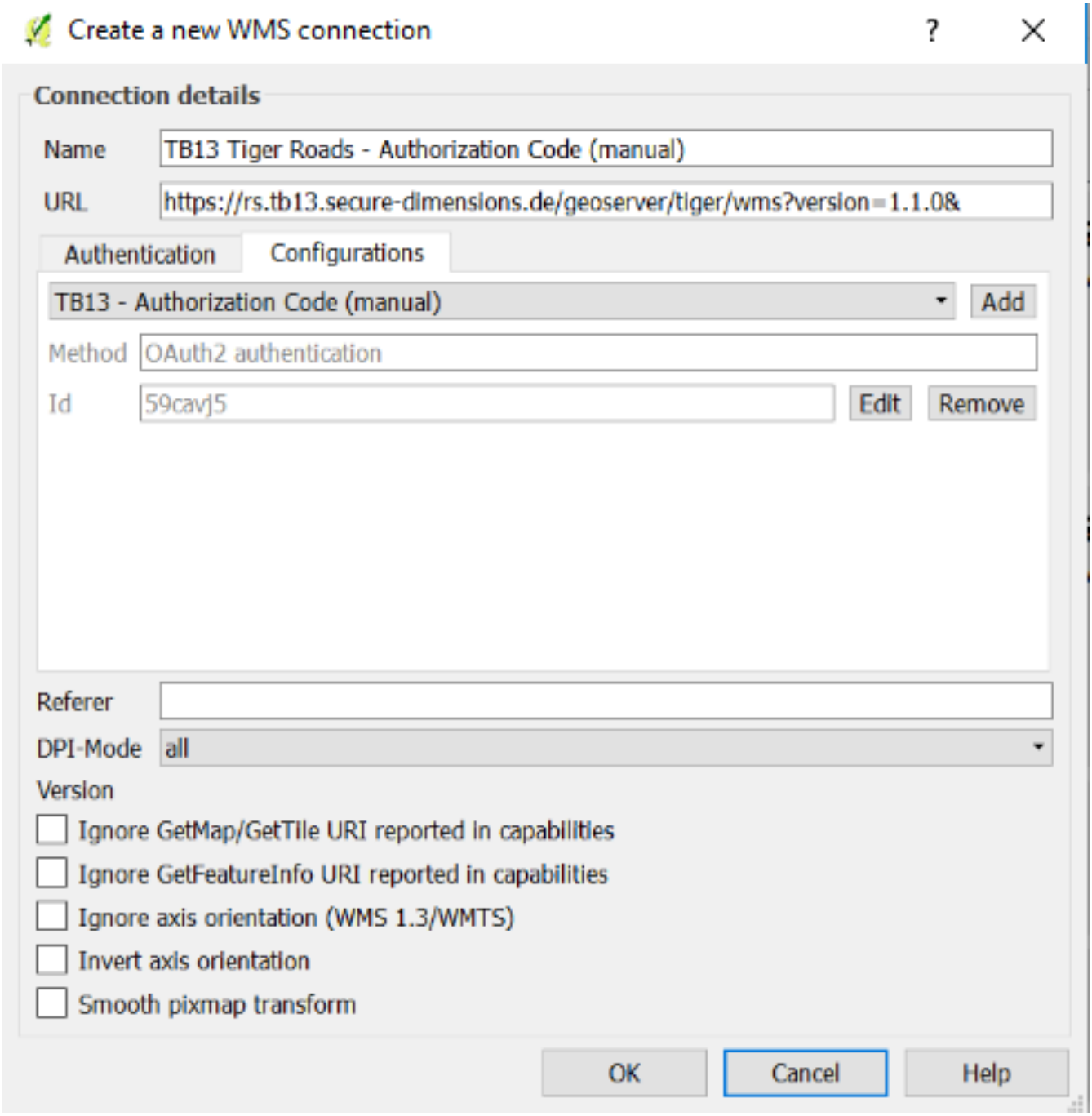


*Figure 35. QGIS configuration*

Figure 35 illustrates the use of the previous OAuth2 configuration for a particular service instance.

After the user connects to the service, the plugin displays an input dialog that requires the user to provide the Authorization Code obtained from the Authorization Server web page. The display of the Authorization Code is the result of the OAuth2 approval the user does in the next steps.

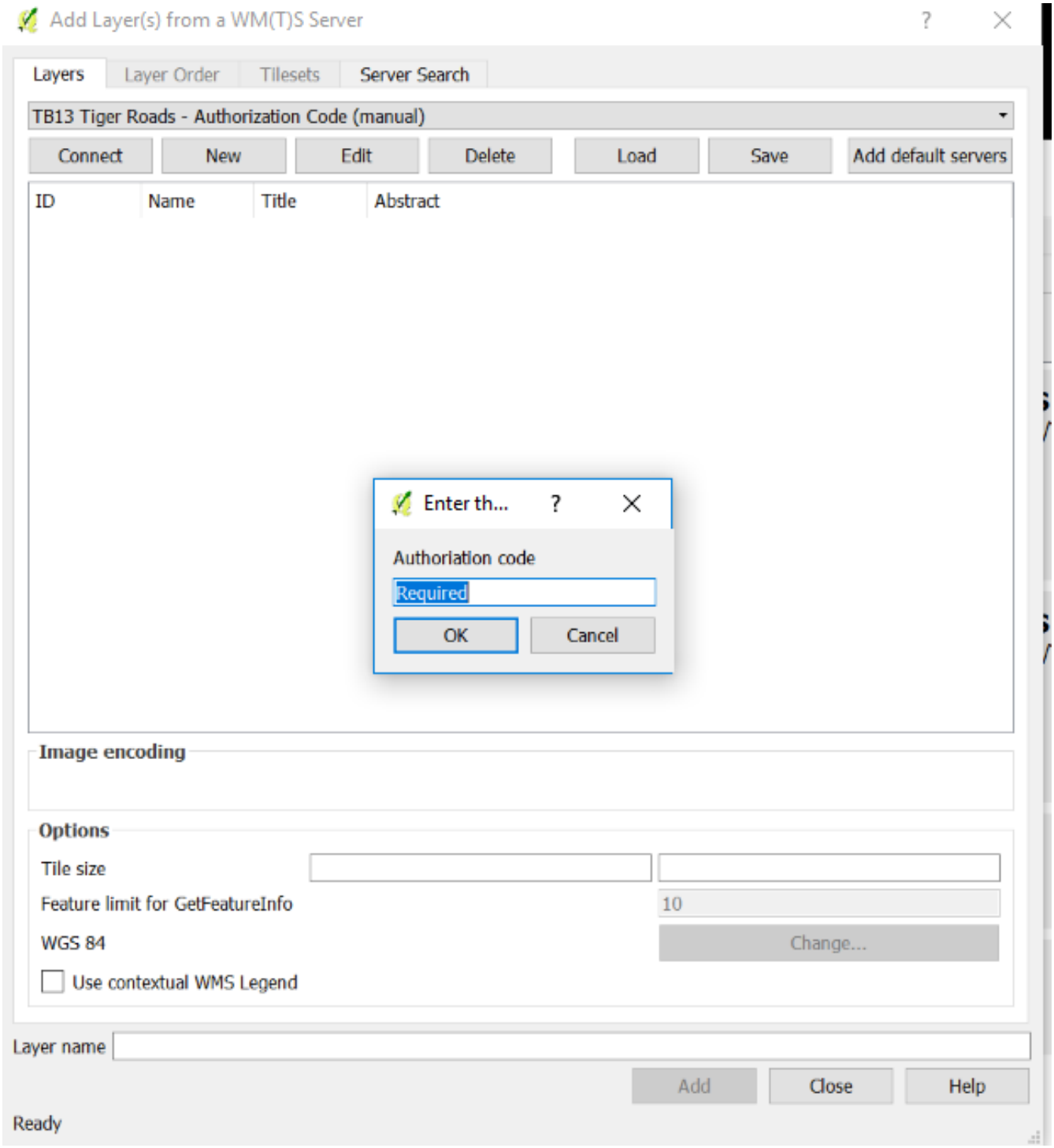


*Figure 36. QGIS Plugin - Authorization Code Input required*

Also after the user connects to the service, the OAuth2 protocol is executed. The first step is for the user to login, as illustrated in Figure 28.

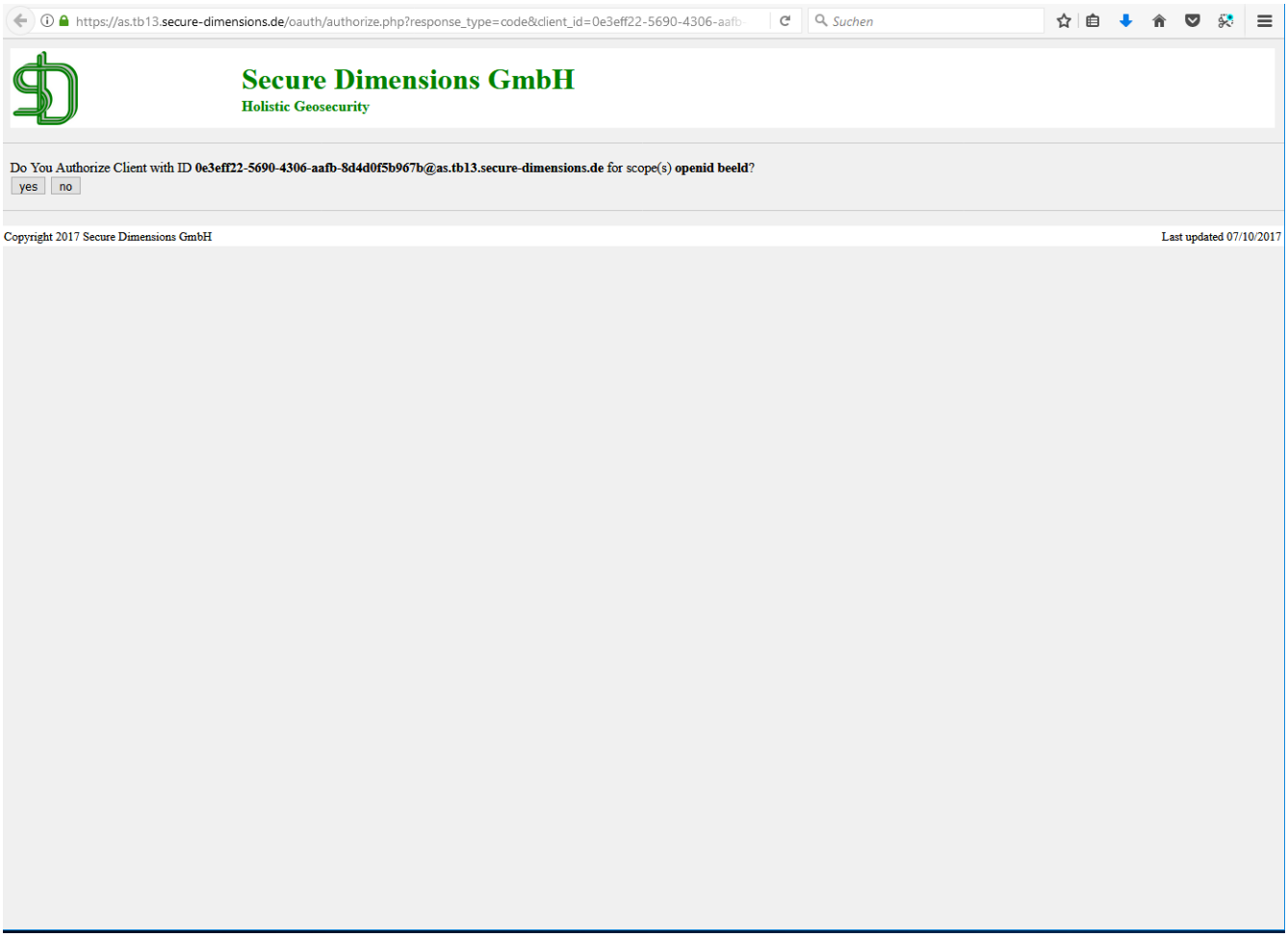


*Figure 37. TB13 Secure Dimensions AS - Approve plugin use*

Before the Authorization Server issues an authorization code, the user must approve this action. The request dialog is illustrated in figure Figure 37.

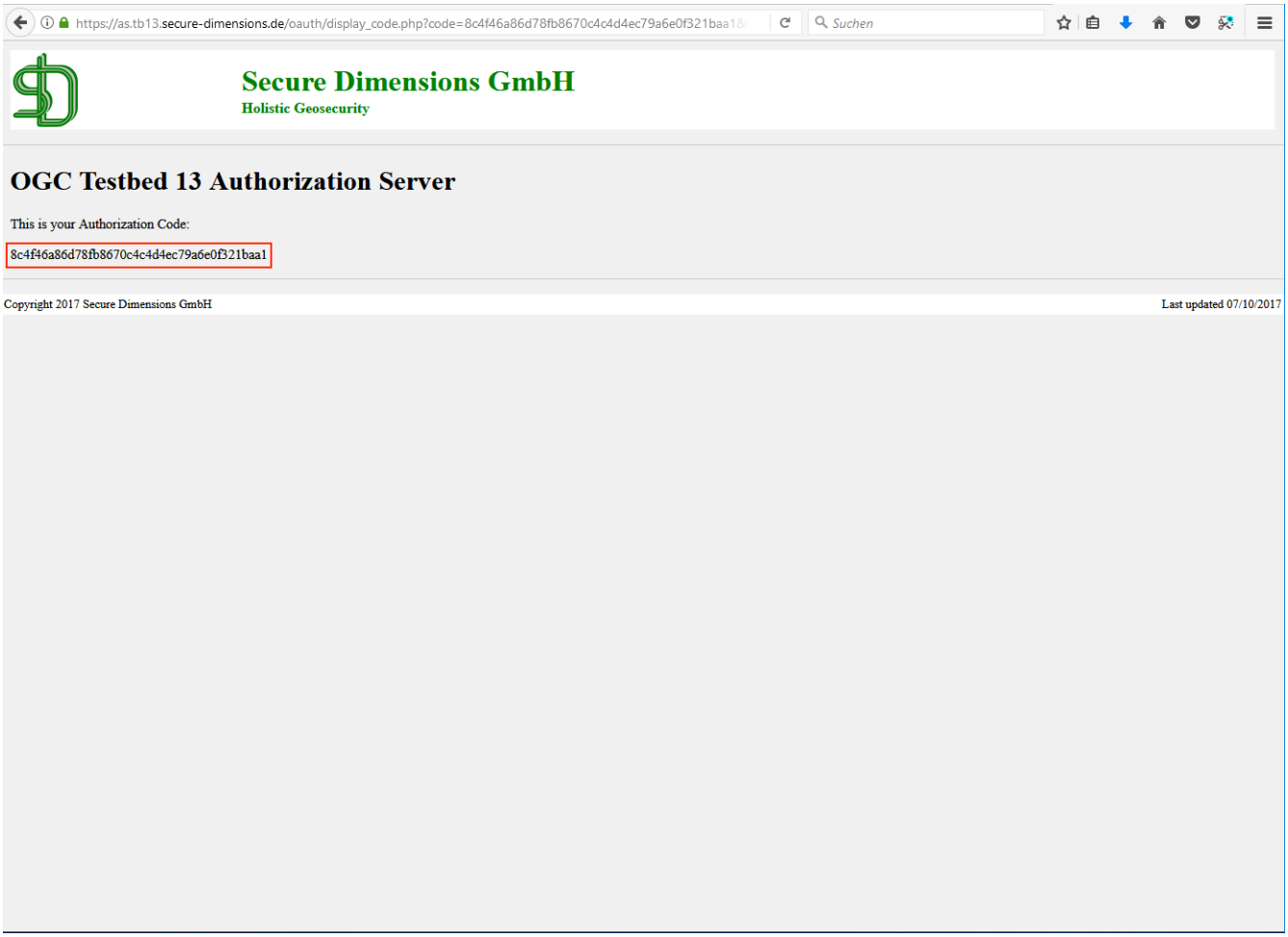


*Figure 38. TB13 Secure Dimensions AS displays the authorization code*

Figure 38 indicates that the Authorization Code Grant with manual has completed successfully. The user has now 10 seconds to copy and paste the code (marked by red rectangle) into the QGIS plugin window (see Figure 36) and click "OK". Assuming the user has done that, QGIS will connect to the service and display the capabilities result.

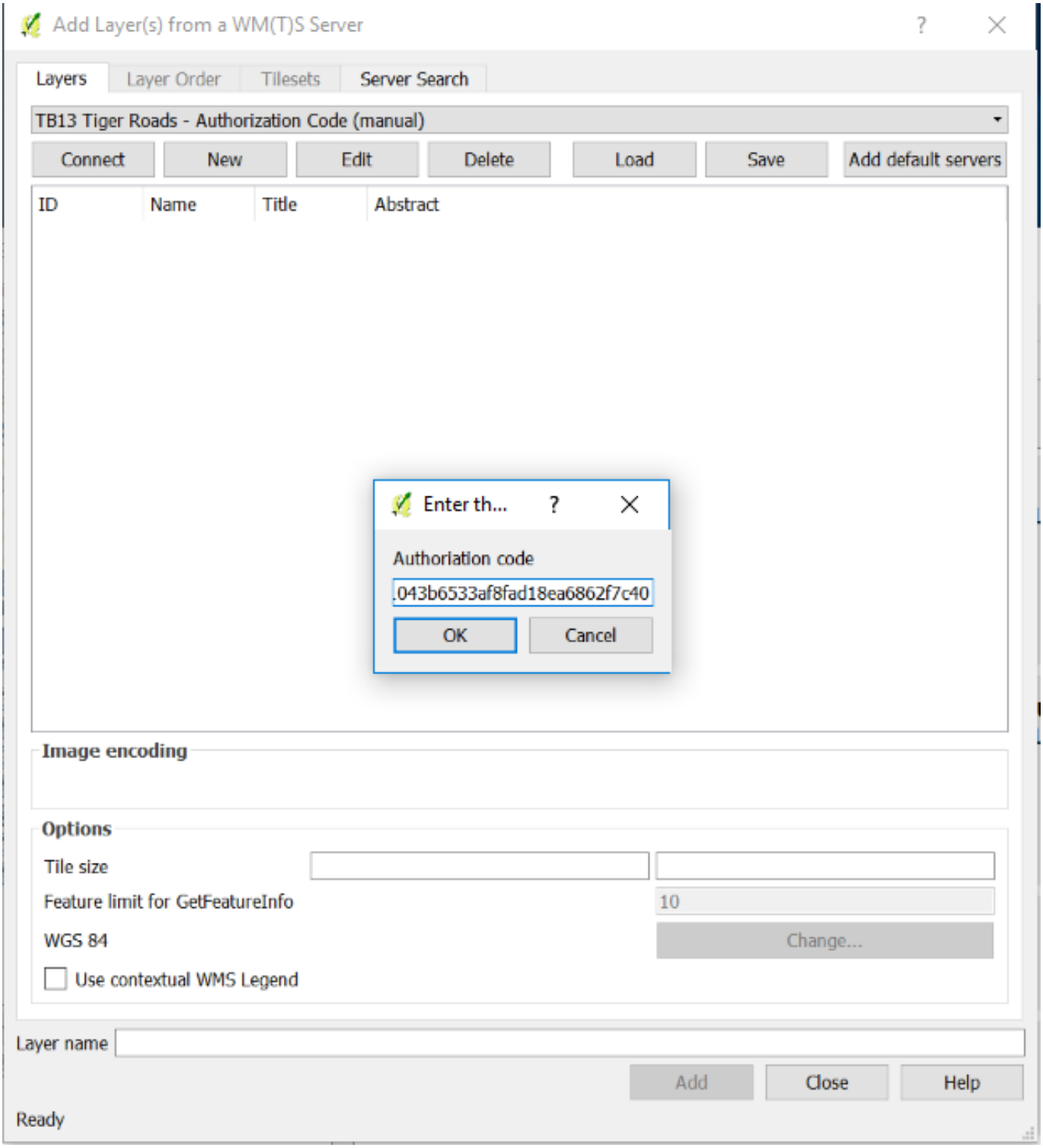


*Figure 39. QGIS user inputs the authorization code*

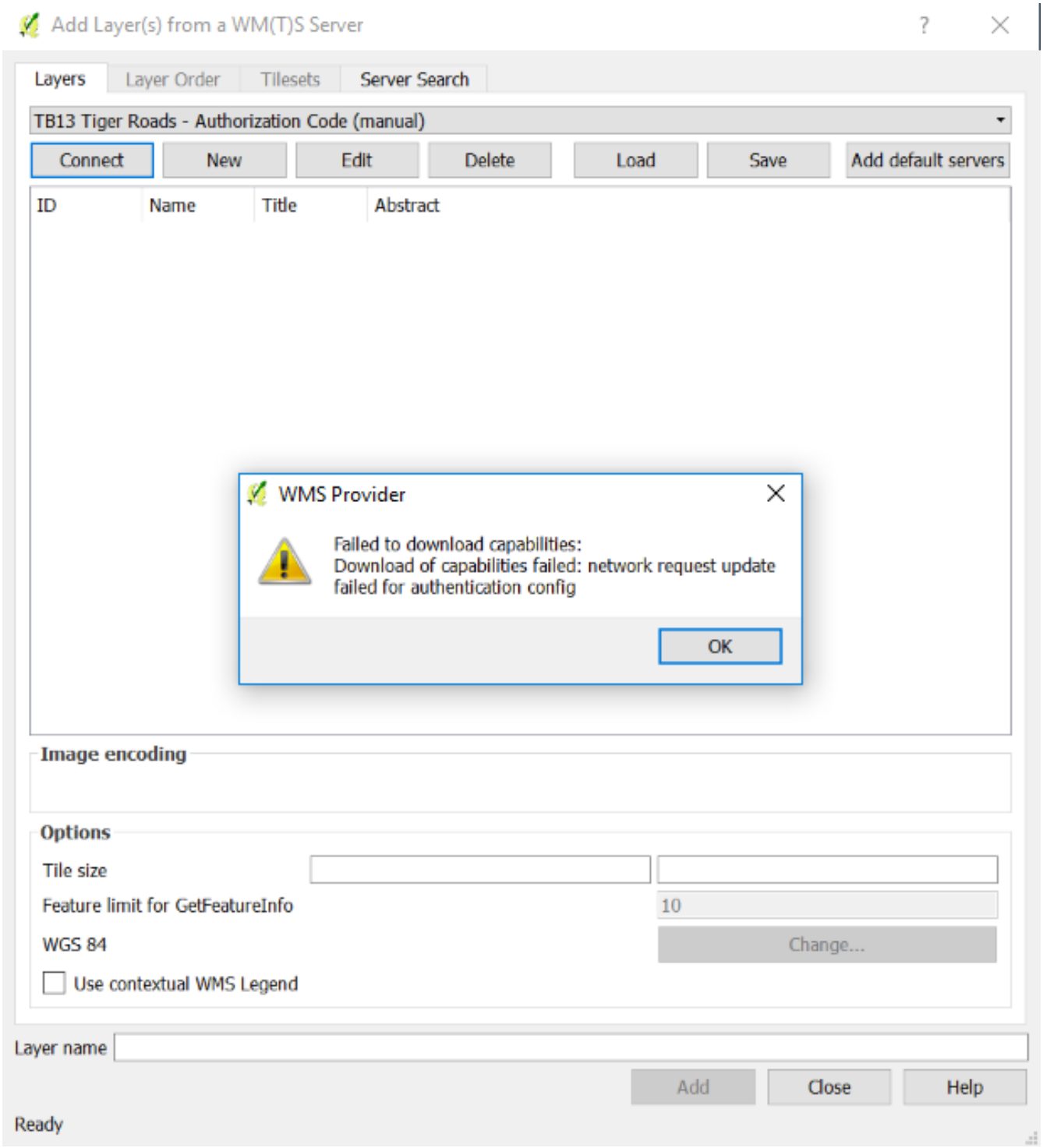


*Figure 40. QGIS user inputs a wrong or expired authorization code*

Figure 40 illustrates the user providing a wrong or expired authorization code.

### 1.1.3. TIE with SECD RS and AS using ROPC

Once the QGIS plugin is registered with the AS, the required parameters must be provided into the plugin GUI.

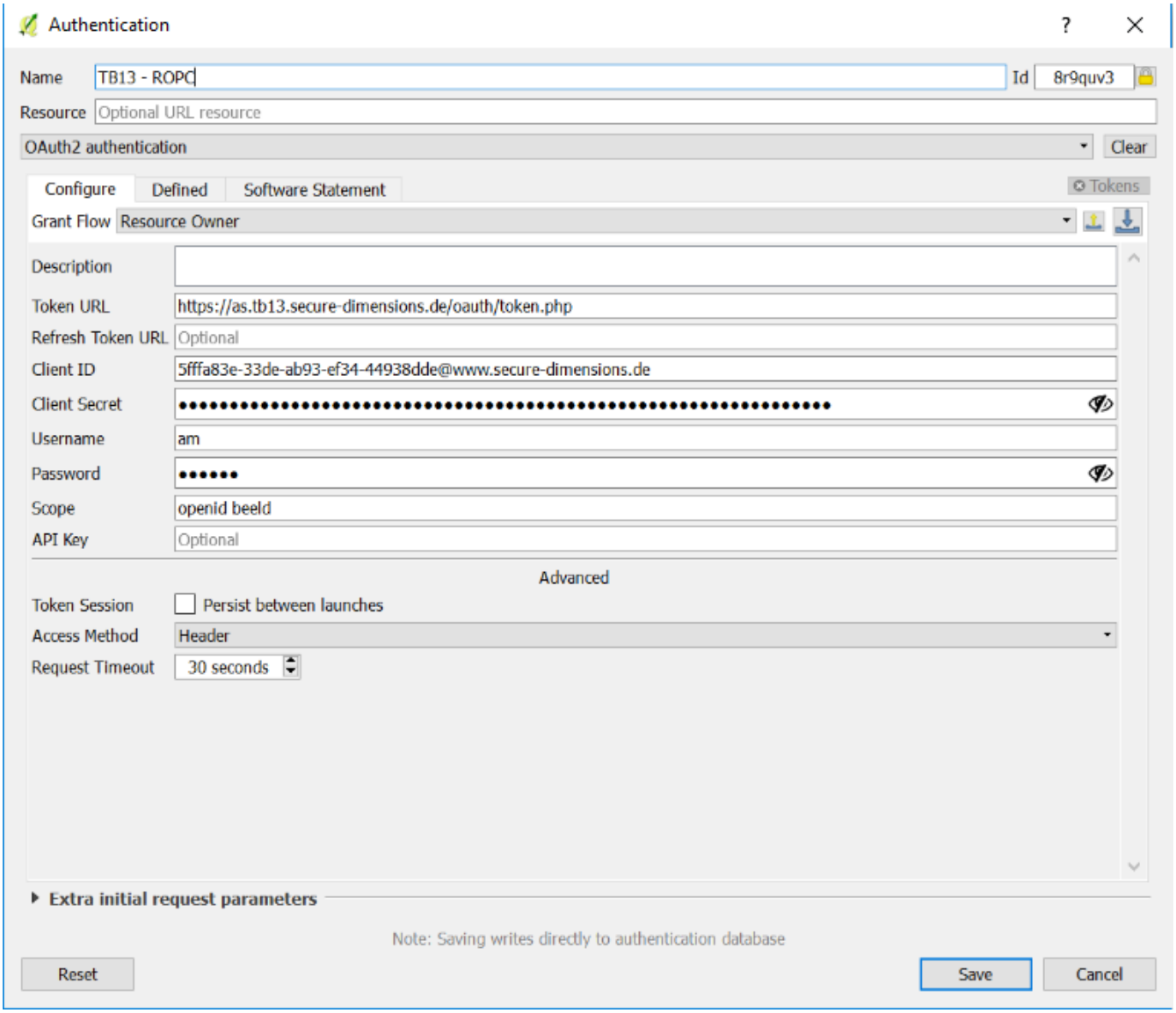


*Figure 41. TB13 Secure Dimensions AS - Resource Owner Password Credentials*

Figure 41 illustrates the configuration of the plugin regarding the TIE as defined in table 1.2. The configuration is saved under the title provided in the name field: "TB13 - ROPC". In addition to the Authorization Server specific endpoints, the configuration indicates that the plugin leverages the Resource Owner Password Credentials and a Resource Server sepcific scope: "beeld". The Client Secret - required for this grant type - is also to be provided for this configuration.

Figure 41 also illustrates that the access token will be attached to the HTTP request leveraging the HTTP header.

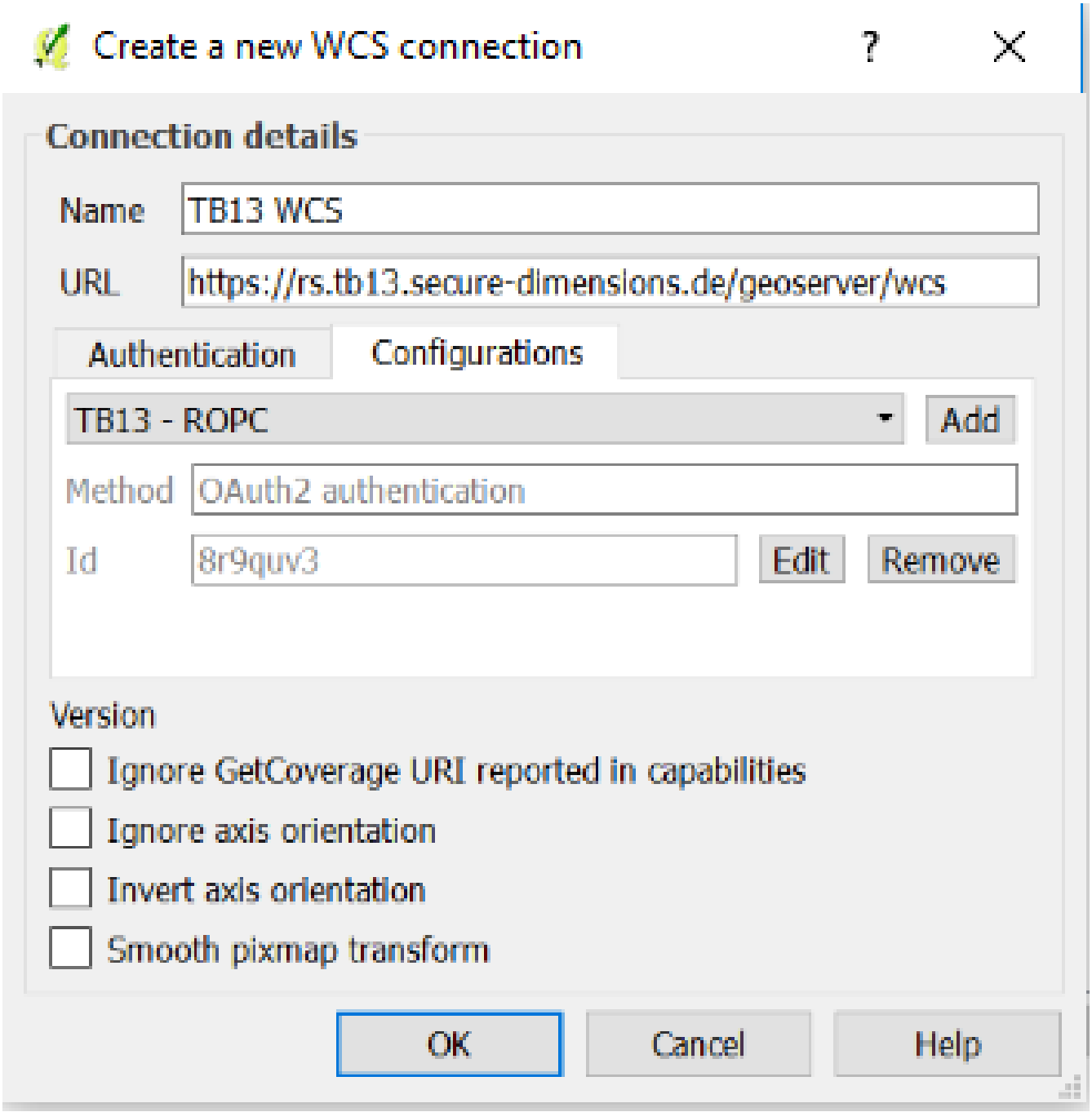


*Figure 42. QGIS configuration*

Figure 42 illustrates the use of the previous OAuth2 configuration for a particular service instance.

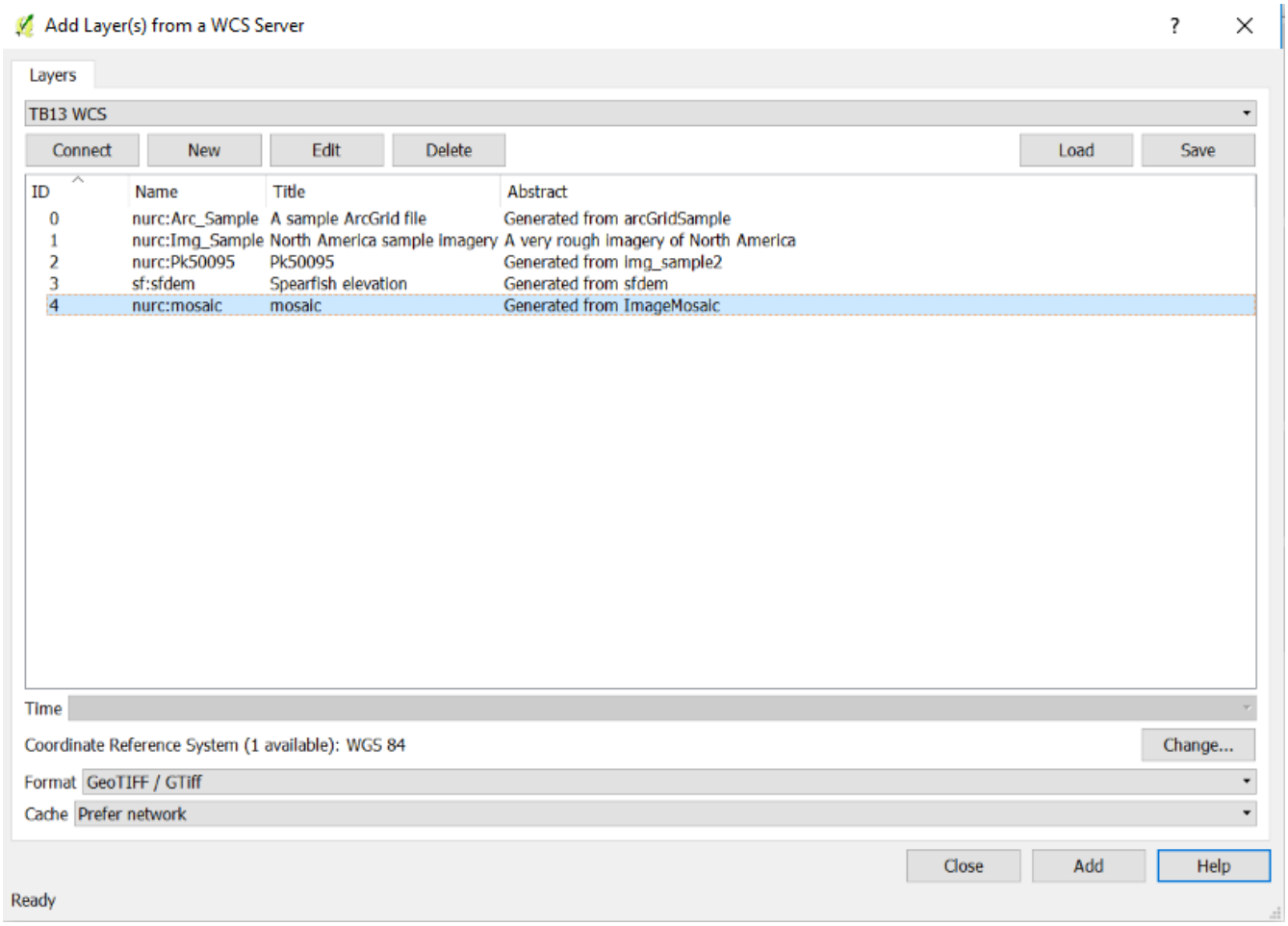


*Figure 43. TB13 Secure Dimensions RS - Capabilities Response*

Once the user returned to the QGIS application, the user must select a WCS layer. For this TIE leveraging the scope "beeld", the layer nurc:mosaic can be selected as illustrated in Figure 43.

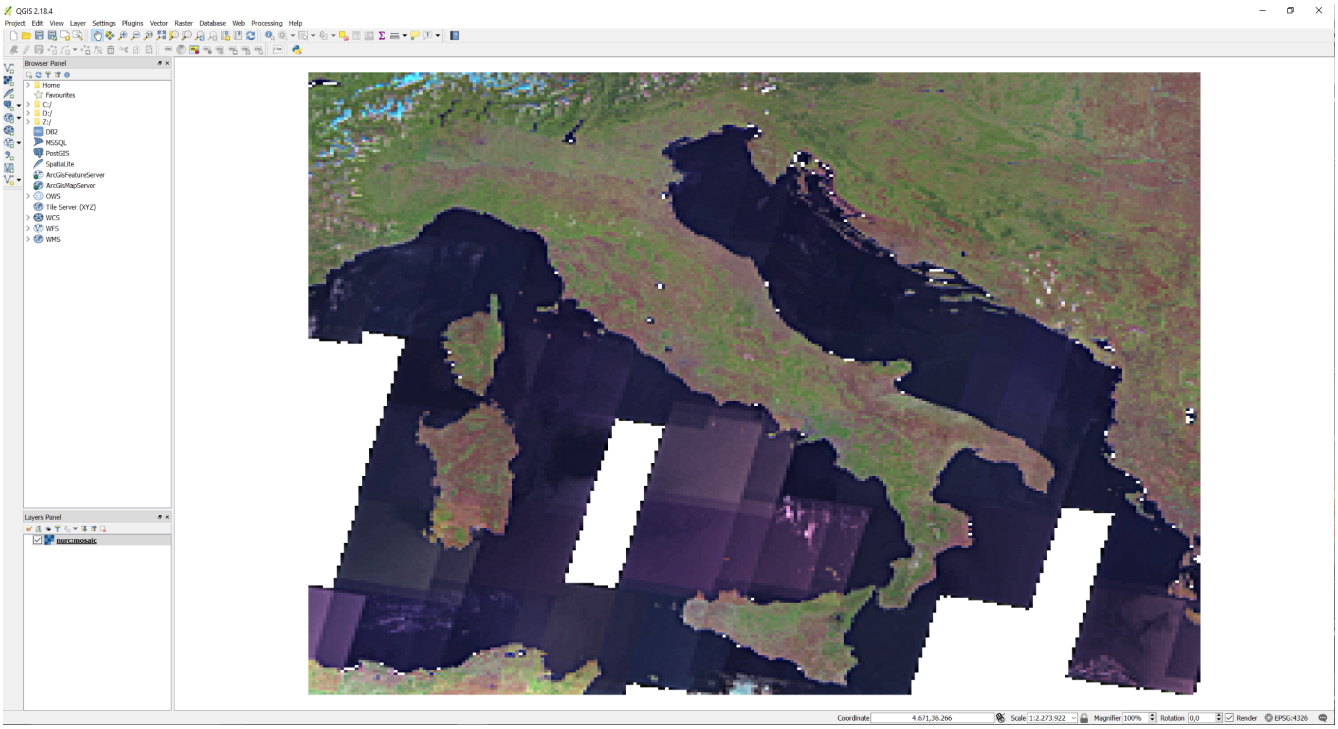

*Figure 44. QGIS displaying map from TB13 Secure Dimensions RS*

After the user has added the layer to the QGIS application, it is loaded and displayed as illustrated

in Figure 44.

*Note*

```
The OAuth2 Resource Owner Password Credentials (ROPC) flow does not require the user
to authorize the plugin. The ROPC is a simplified direct flow where the user is not
active.
```

### 1.1.4. TIE with Dutch Kadaster RS and AS

Once the QGIS plugin is registered with the AS, the required parameters must be provided into the plugin GUI.

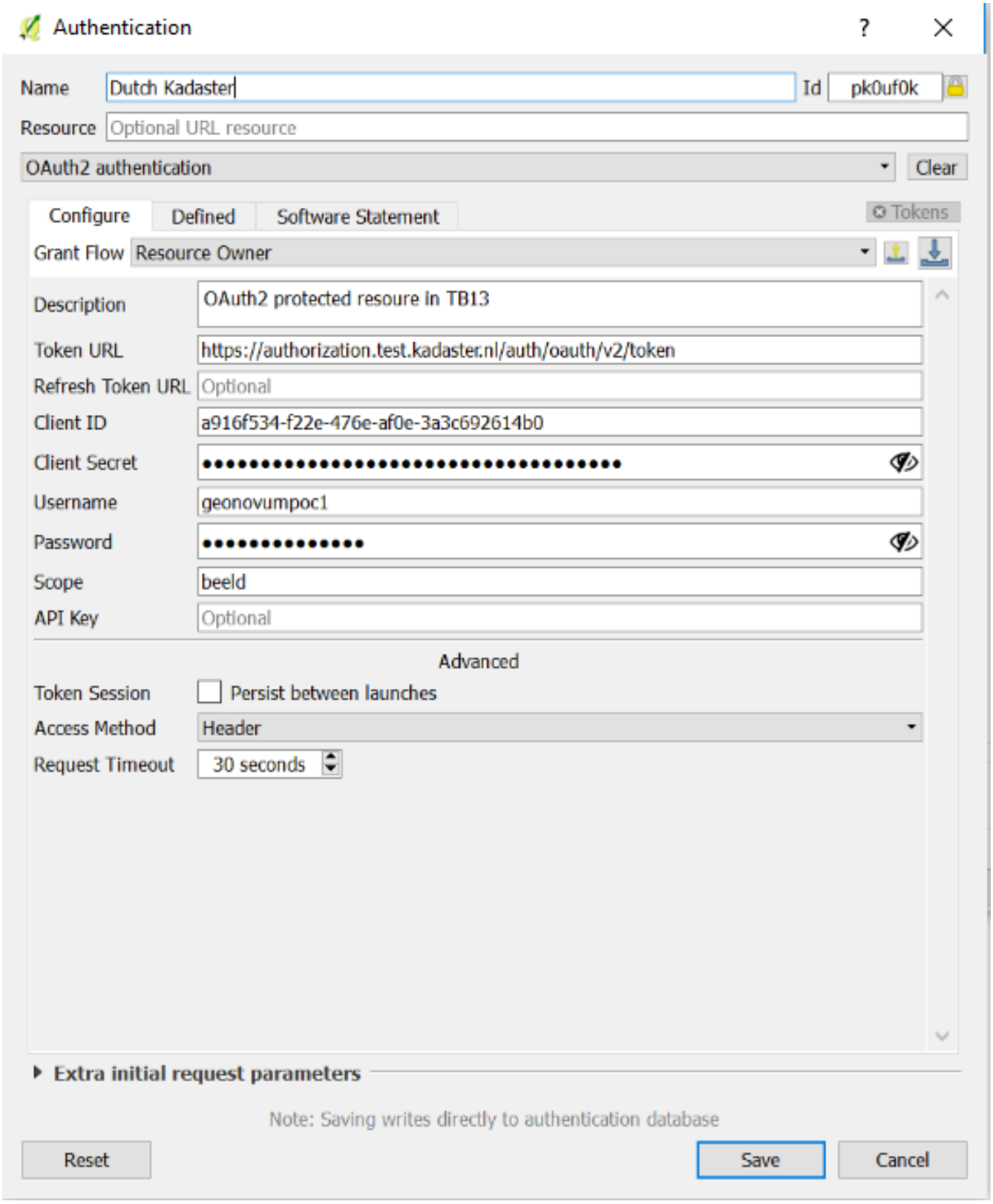


*Figure 45. TB13 Dutch Kadaster AS - Resource Owner Password Credentials*

Figure 45 illustrates the configuration of the plugin regarding the TIE as defined in table 1.3. The configuration is saved under the title provided in the name field: "Dutch Kadaster - ROPC". In addition to the Authorization Server specific endpoints, the configuration indicates that the plugin leverages the Resource Owner Password Credentials and a Resource Server sepcific scope: "beeld". The Client Secret - required for this grant type - is also to be provided for this configuration.

Figure 45 also illustrates that the access token will be attached to the HTTP request leveraging the HTTP header.

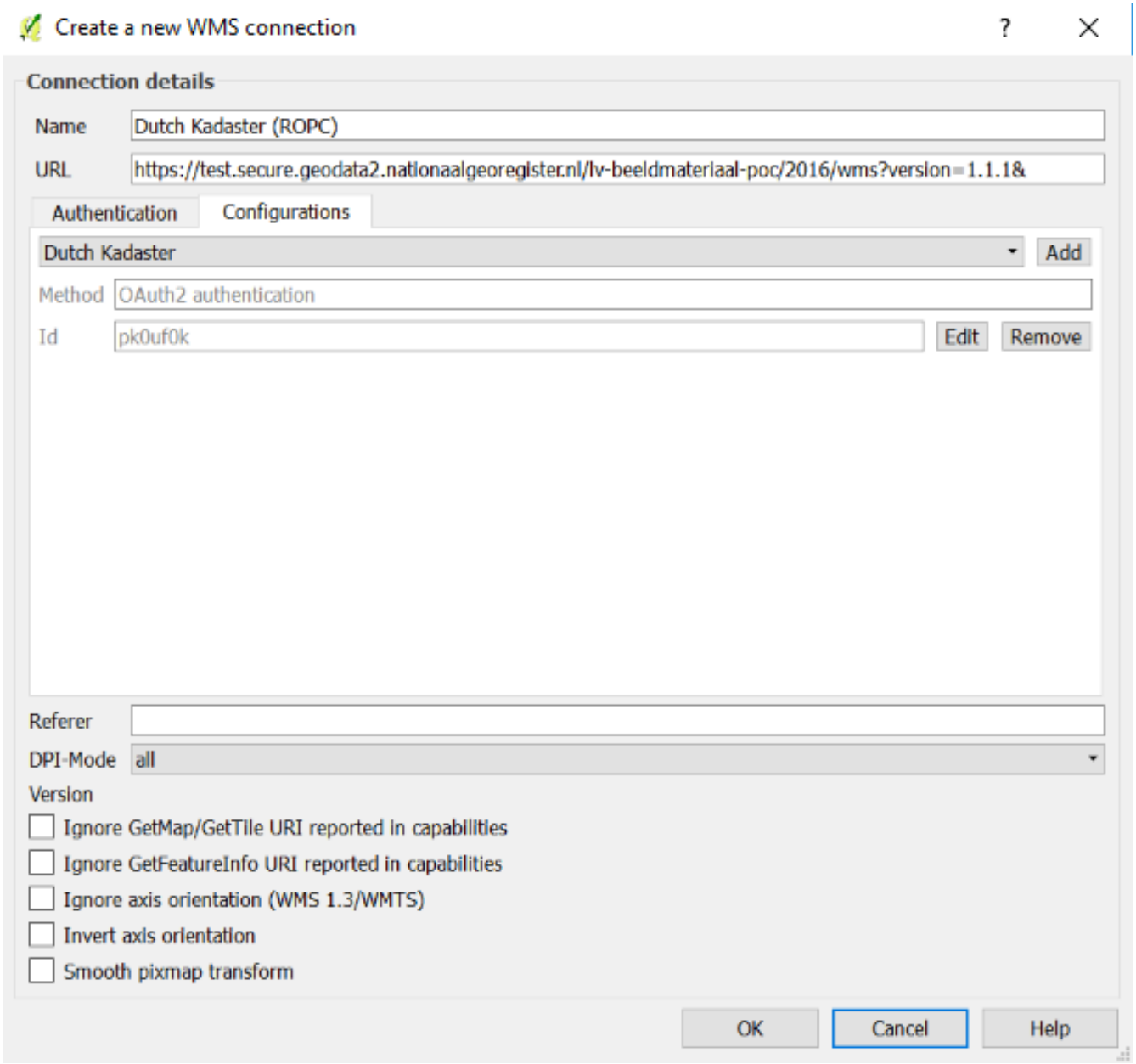


*Figure 46. QGIS configuration*

Figure 46 illustrates the use of the previous OAuth2 configuration for a particular service instance.

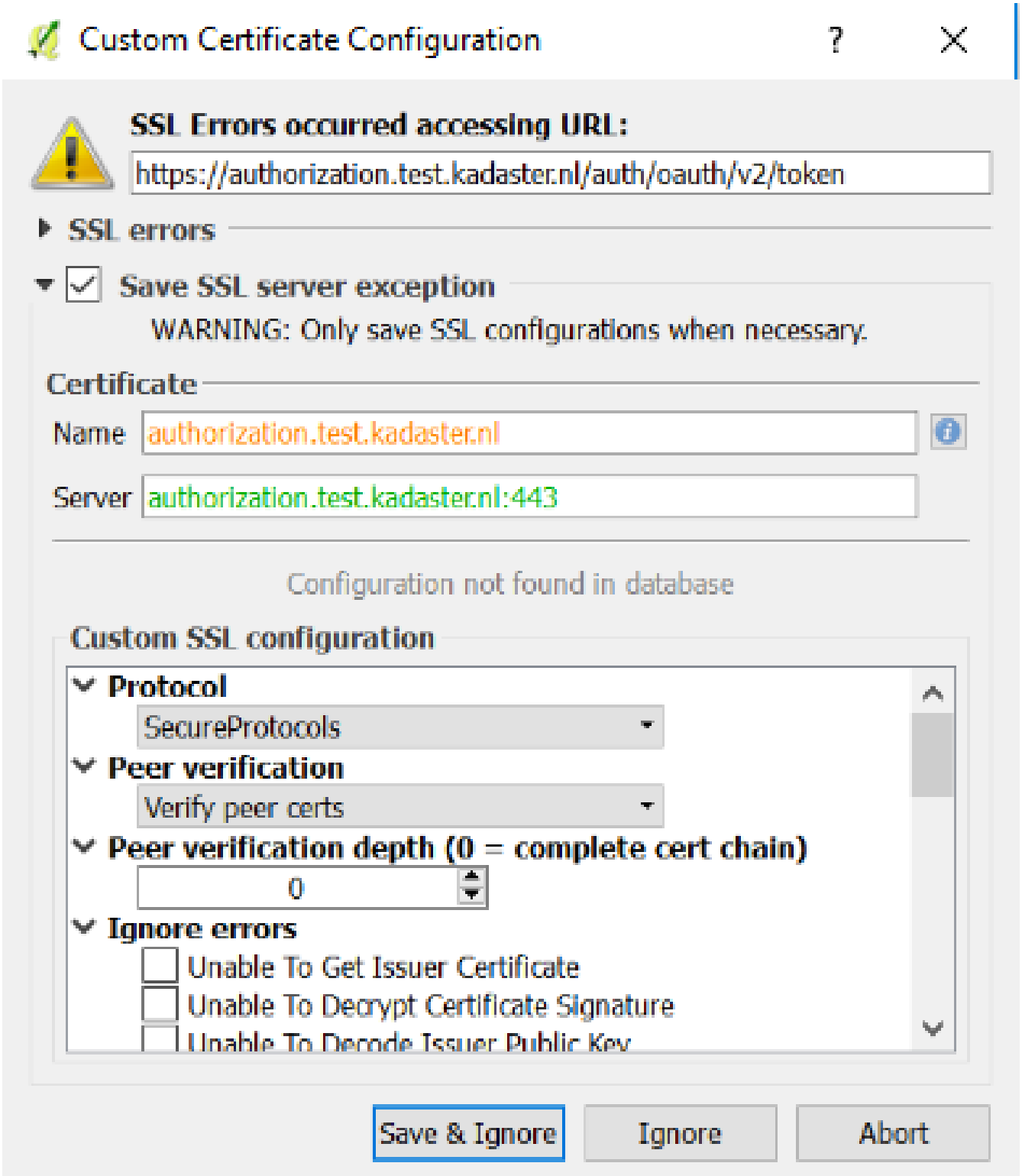


*Figure 47. TB13 Dutch Kadaster RS - SSL Certificate Warning*

Figure 47 illustrates the QGIS application showing an SSL certificate warning, as the certificate in place is a self-signed test certificate. Therefore, this user warning dialog is very appropriate.

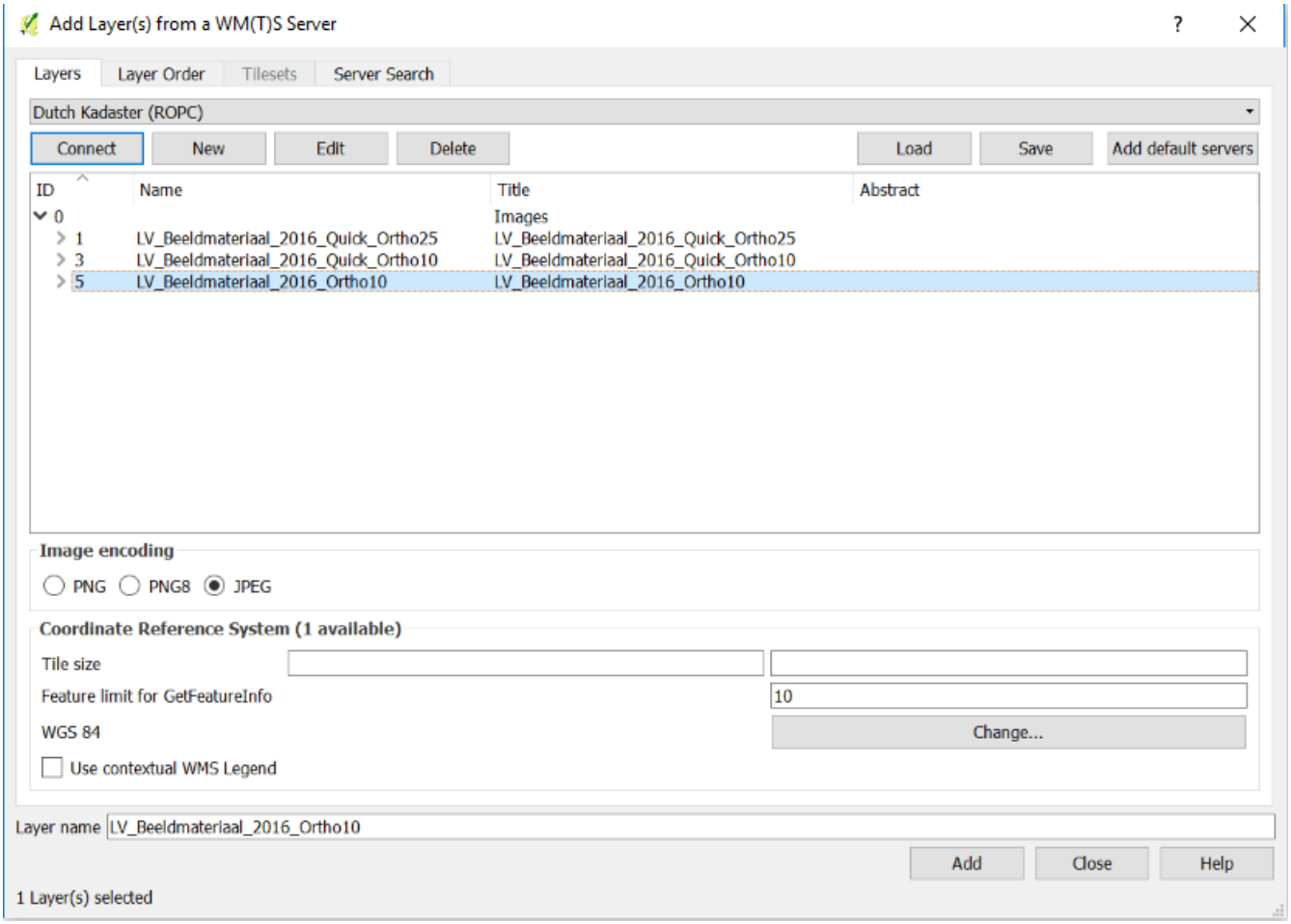


*Figure 48. TB13 Dutch Kadaster RS - Capabilities Response*

Once the user has returned to the QGIS application, the user must select a WMS layer. For this TIE leveraging the scope "beeld", any orthophoto layer can be selected as illustrated in Figure 48.

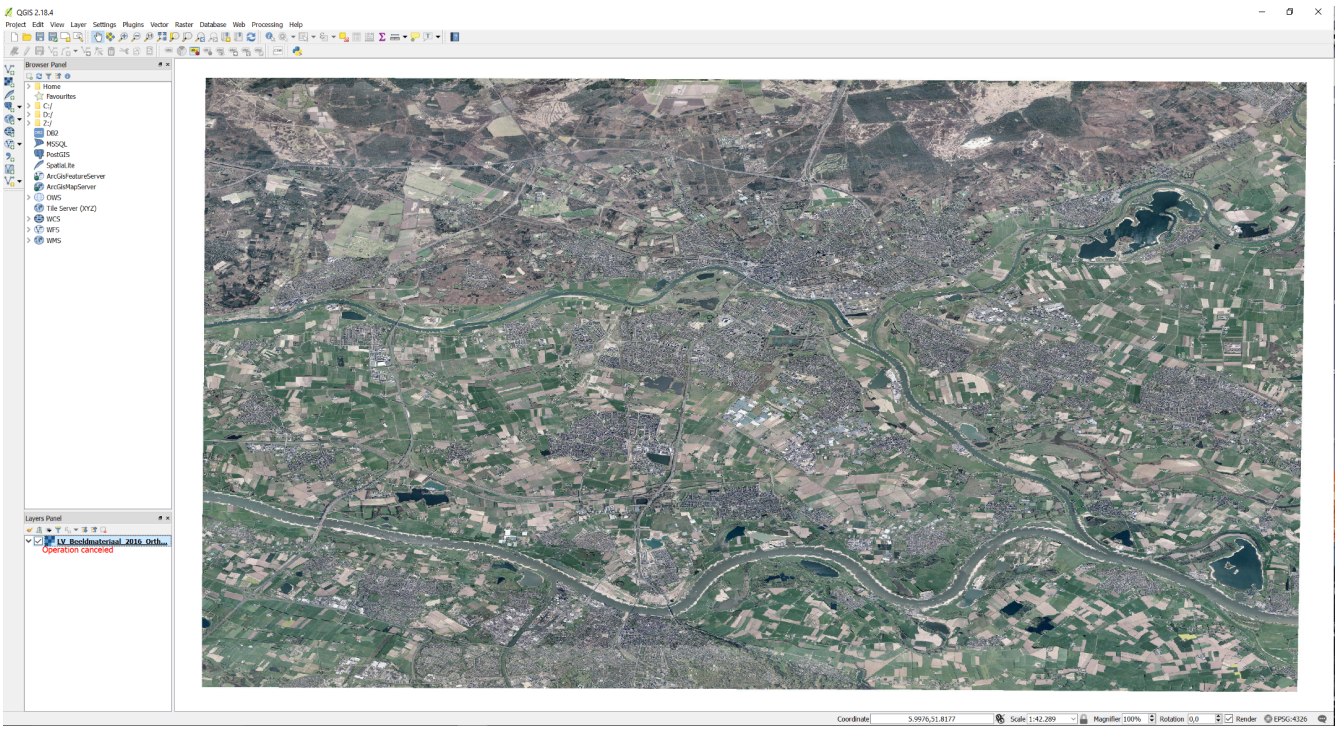

*Figure 49. QGIS displaying map from TB13 Dutch Kadaster RS*

After the user has added the layer to the QGIS application, it is loaded and displayed as illustrated

in Figure 49.

*Note*

```
The OAuth2 Resource Owner Password Credentials (ROPC) flow does not require the user
to authorize the plugin. The ROPC is a simplified direct flow where the user is not
active.
```

### 1.1.5. TIE with 52North WFS and Auth0.com AS

Once the QGIS plugin is registered with the AS, the required parameters must be provided into the plugin GUI.

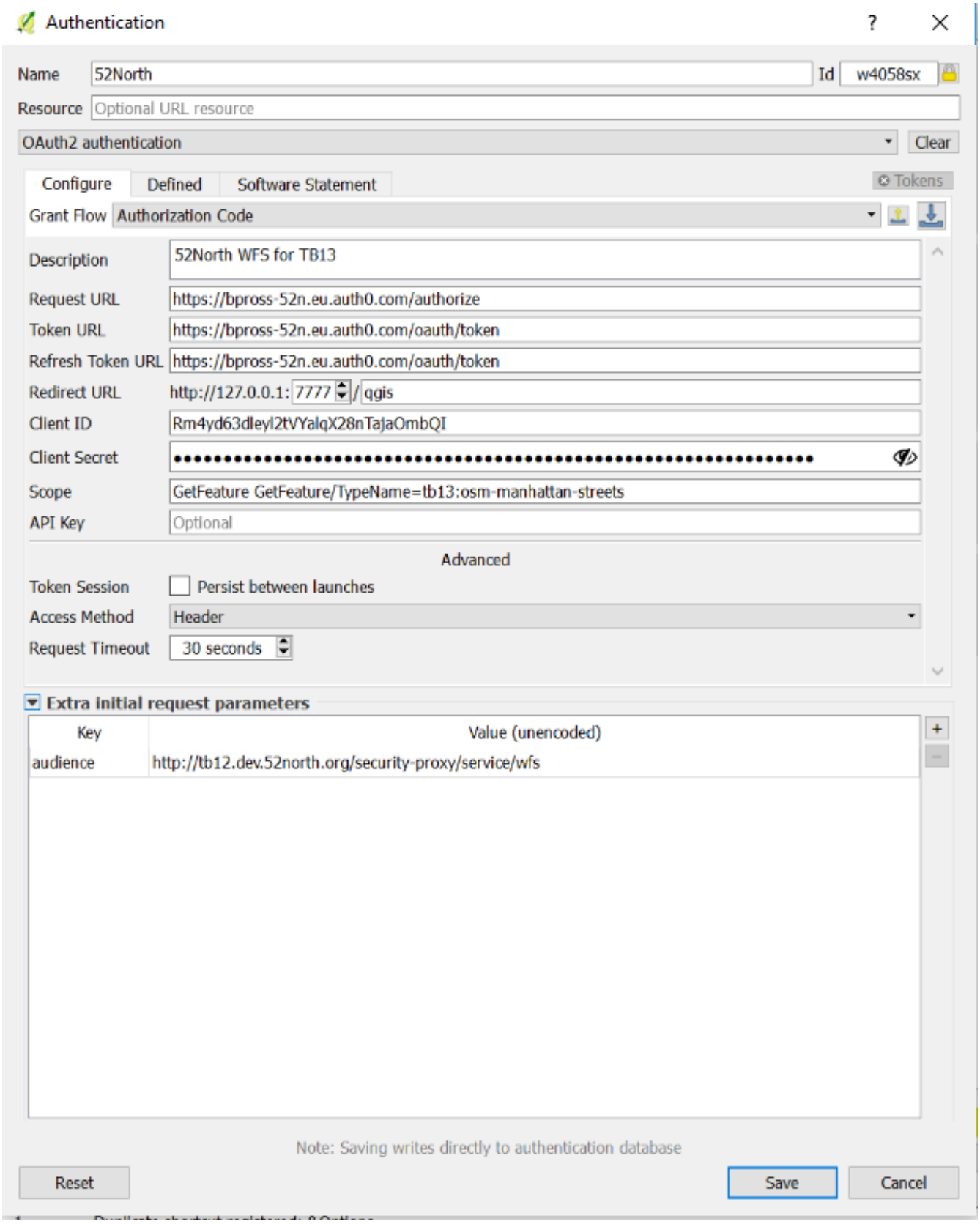


*Figure 50. TB13 Auth0.com AS - Authorization Code Grant*

Figure 50 illustrates the configuration of the plugin regarding the TIE as defined in table 1.4. The configuration is saved under the title provided in the name field: "52 North". In addition to the Authorization Server specific endpoints, the configuration indicates that the plugin leverages the Authorization Code Flow Grant and a Resource Server specific audience: "http://tb12.dev.52north.org/security-proxy/service/wfs".

Figure 50 also illustrates that the access token will be attached to the HTTP request leveraging the HTTP header.

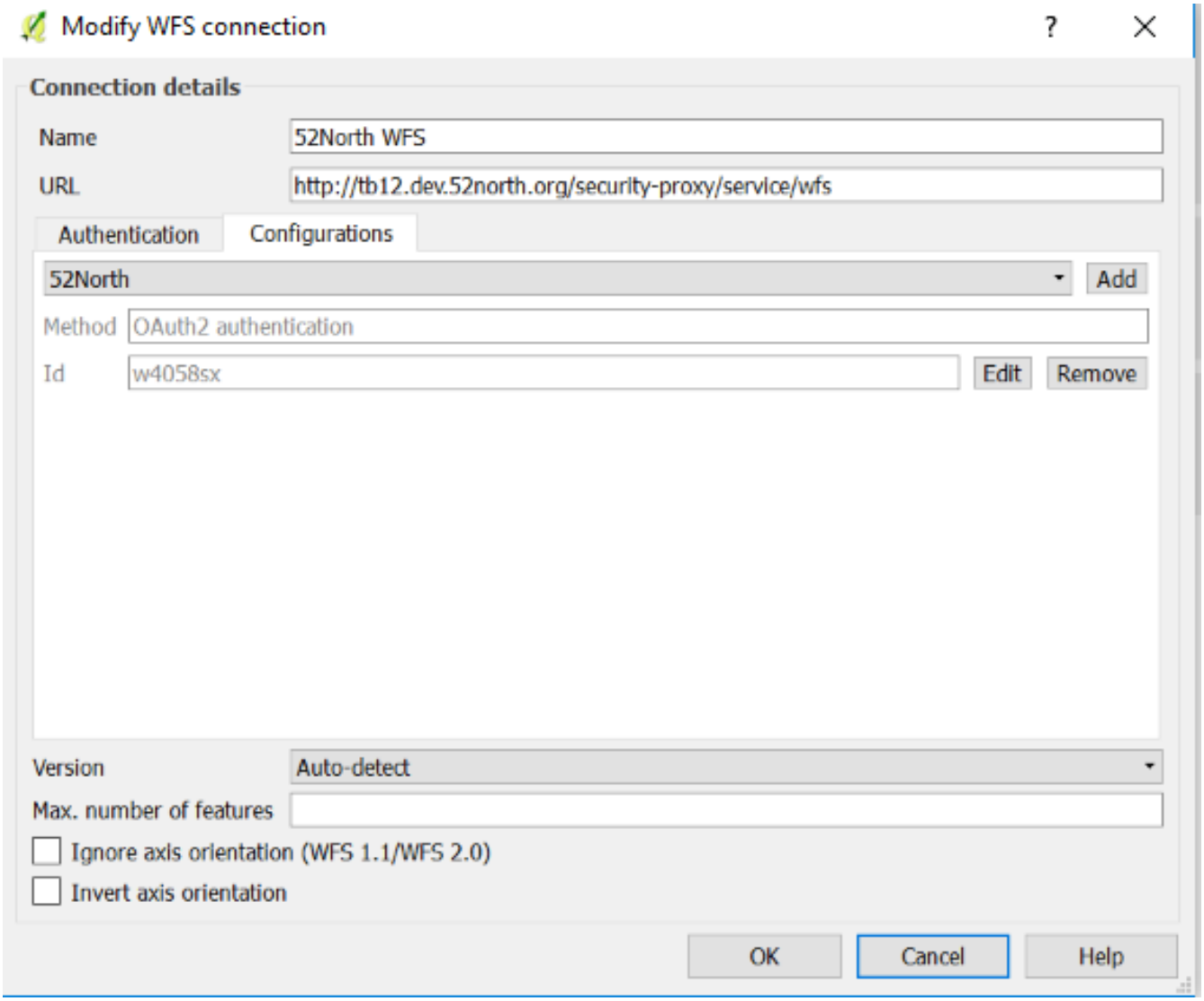


*Figure 51. QGIS configuration*

Figure 51 illustrates the use of the previous OAuth2 configuration for a particular service instance.

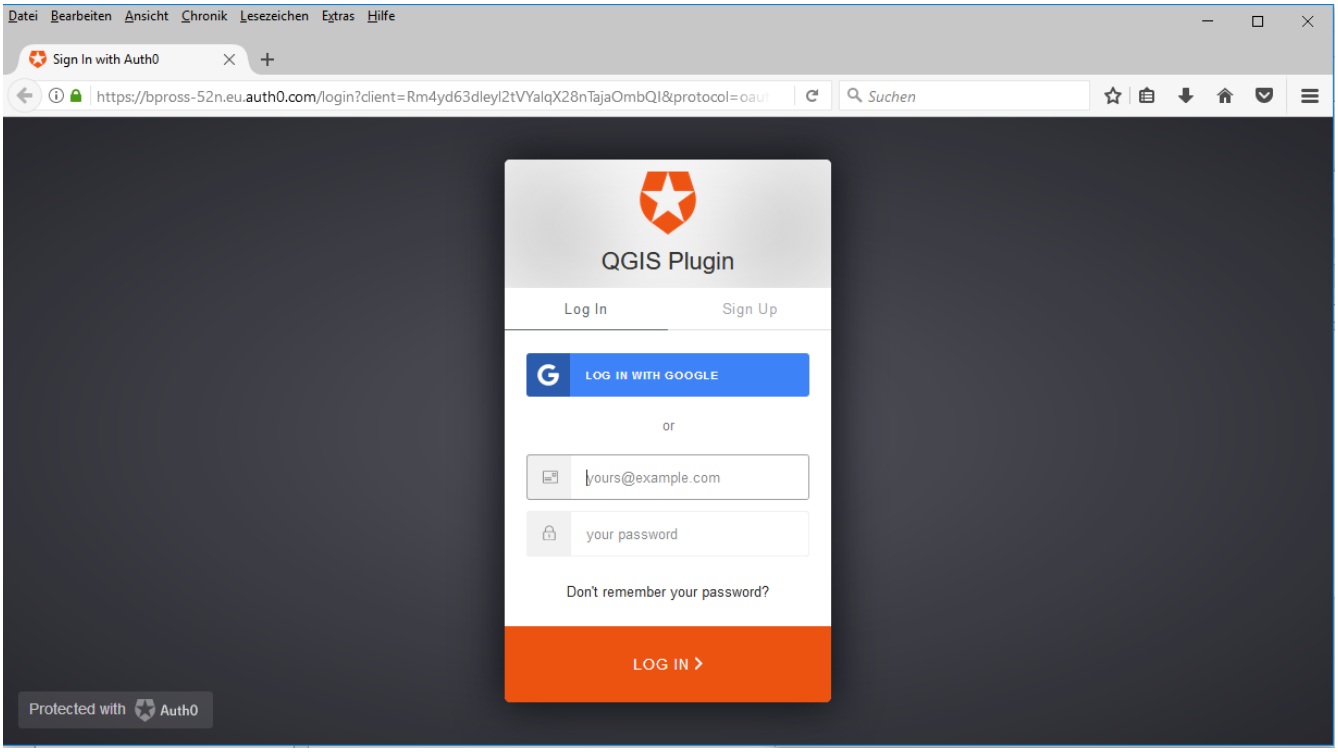


*Figure 52. TB13 Auth0.com AS - Login*

After the user connects to the service, the OAuth2 protocol is executed. The first step is for the user to login, as illustrated in Figure 52.

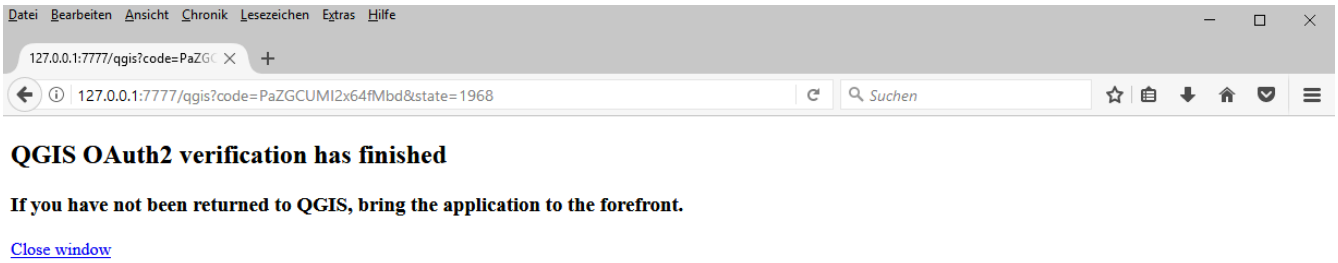


*Figure 53. Local browser - TB13 Auth0.com AS approval finished*

Figure 53 indicates the Authorization Code Grant has successfully completed with the Authorization Server. The user is informed to close the window (which might not work depending on the Web Browser and other configuration parameters) or focus to the QGIS application.

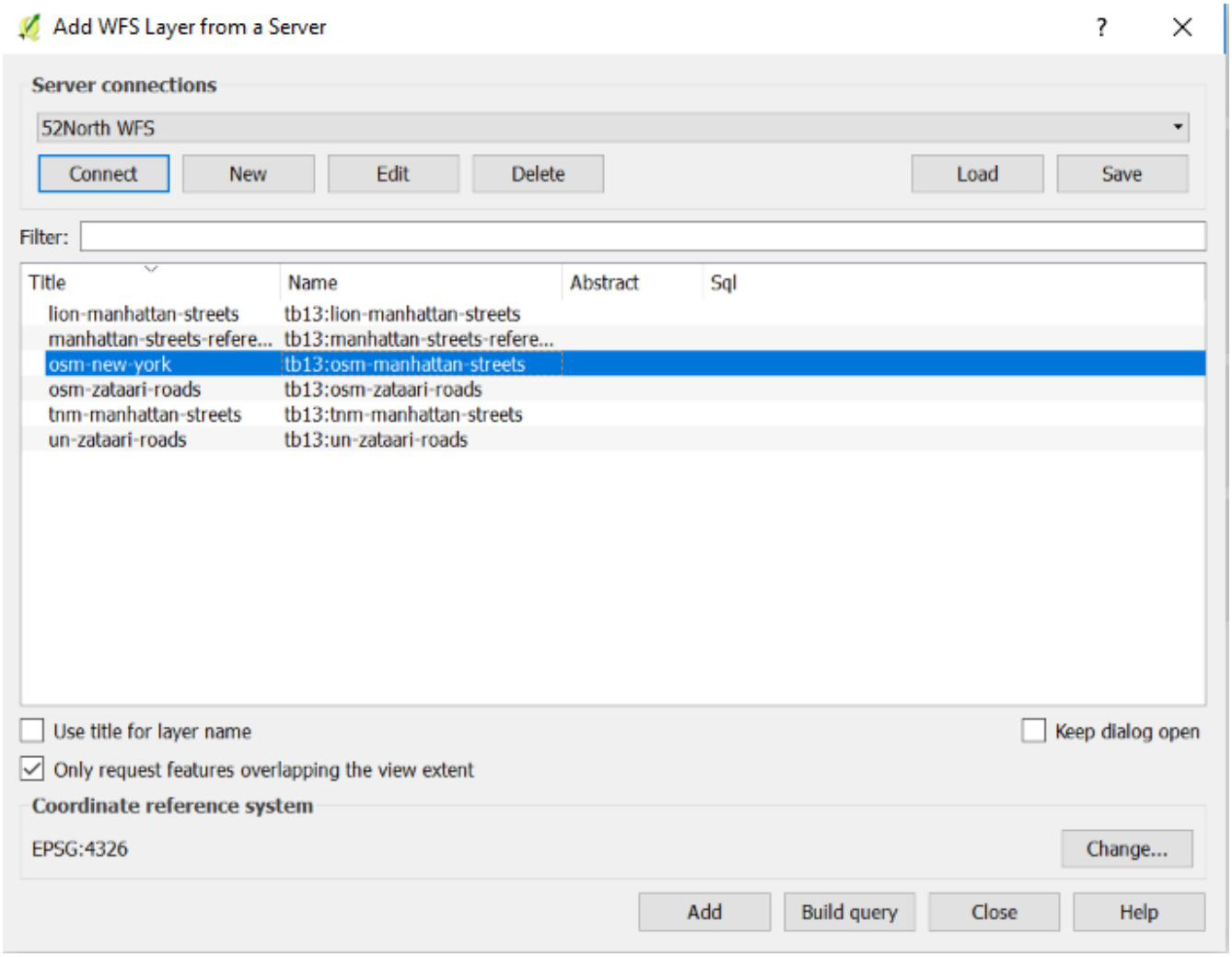


*Figure 54. TB13 52North RS - Capabilities Response*

Once the user returned to the QGIS application, the user must select a WFS feature type. For this TIE leveraging the scopes "GetFeature GetFeature/TypeName=tb13:osm-manhattan-streets", the feature type tb13:osm-manhattan-streets can be selected as illustrated in Figure 54.

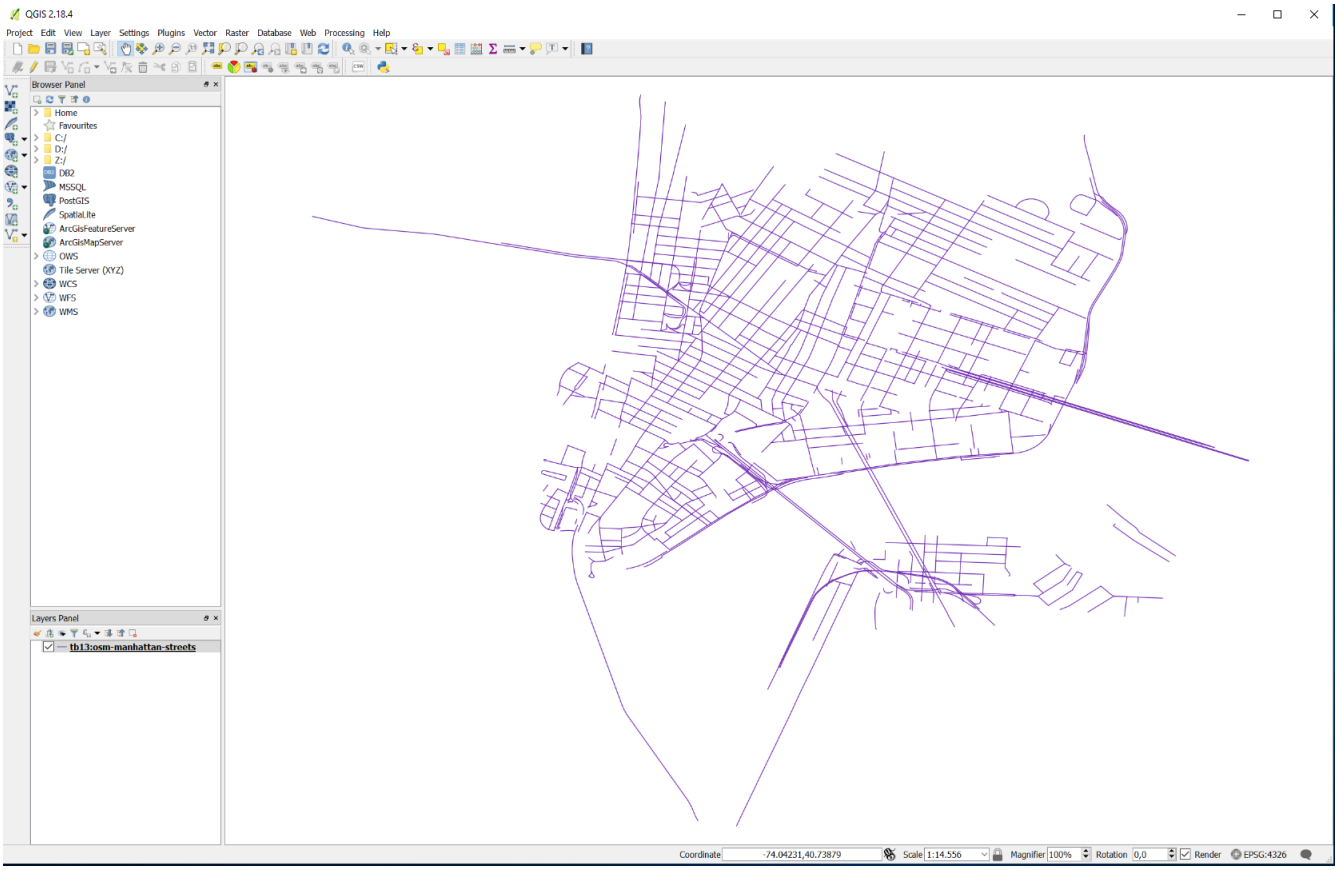


*Figure 55. QGIS displaying feature collection from TB13 52North RS*

After the user has added the feature type to the QGIS application, it is loaded and displayed as illustrated in Figure 55.

# Appendix B: How to use the QGIS Plugin for OAuth2

This section provides information for the end user of the QGIS plugin covering the installation and use of the plugin.

## 1.1. Installation

The QGIS Plugin provided under OGC Testbed 13 is compiled as a Windows 32bit executable to work with QGIS 32bit build 2.18.4. However, the compiled DLL (oauth2authmethod.dll) does also work with the QGIS 2.18.13 version.

The actual installation of the plugin is quite simple: You just download the plugin from this URL (https://as.tb13.secure-dimensions.de/oauth2authmethod.dll) and put the DLL into the QGIS "plugins" directory. This is usually located under the diretory for the QGIS installation.

*Note*

```
Assuming your QGIS installation is in C:\Program Files\QGIS-2.18.13 then you must copy
the QGIS plugin DLL (oauth2authmethod.dll) into the directory C:\Program Files\QGIS-
2.18.13\apps\qgis\plugins
```

## 1.2. Compile from Sources

For Testbed 13, Secure Dimensions provides a compiled version of the plugin for easy installation (see previous section). But, it is also possible to compile from the sources. This procedure is more difficult (and only suggested for decent QGIS developers) as you need to build a couple of dependencies. It is not our in tension to provide a step-by-step instructions list, but rather provide an overview so that you can go into the details yourself. For that purpose, we describe the build process based on QGIS version 2.18.4 (which was the latest available at the time of B13 start) and a Windows 10 32bit installation.

First, you need to install Microsoft Visual Studio 2010.

Second, you need to build QGIS 2.18.14. How to do this, please follow the detailed description available from OSGeo: https://github.com/qgis/QGIS/blob/release-2_18/INSTALL

Third, you need to obtain the sources for the QGIS OAuth2 Plugin (master) from Github:

https://github.com/securedimensions/QGIS-OAuth2-Plugin

Then, start cmake and use the plugin installation directory as source. You need to create a build directory and use that with cmake. Then select the compiler based on Visual Studio 2010 (use native compiler). The following figure illustrates the CMake configuration. Please make sure to uncheck the "build test app" option.

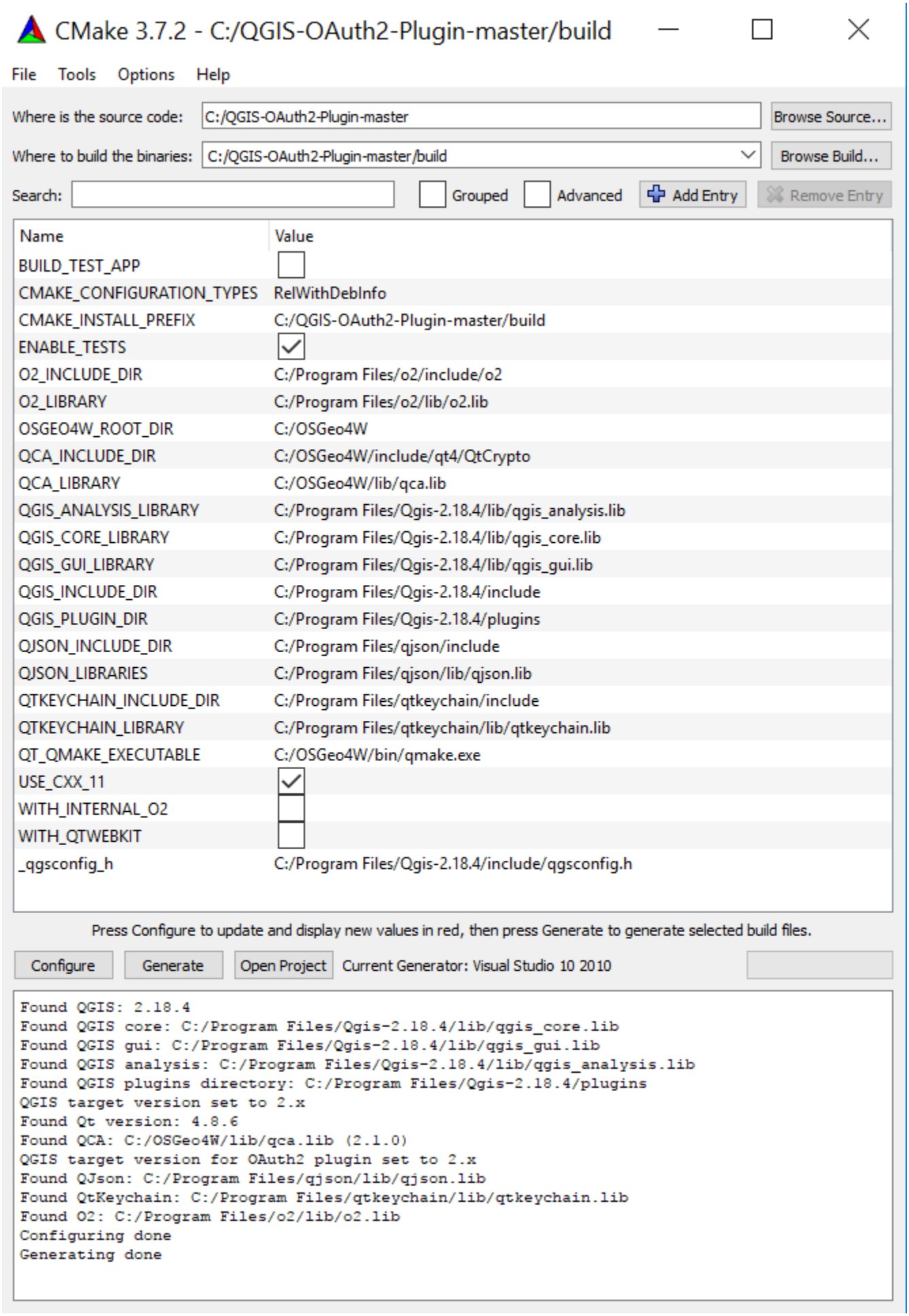


*Figure 56. CMake configuration for building the OAuth2 Plugin for QGIS 2.18.4*

As you can see, some dependencies exist to the OSGeo installation of QGIS (C:/OSGeo4W and C:/Program Files/Qgis-2.18.4/include/qgisconfig.h). But there are other dependencies that need to be built individually:

- O2 library (https://github.com/pipacs/o2)
- QJSON library (http://qjson.sourceforge.net/)
- qtkeychain library (https://github.com/frankosterfeld/qtkeychain)

Final note: In order for the installation of the DLL to finish with no errors, it is important that you change the permissions on the target directory to allow any access: C:/Program Files/Qgis-2.18.4/plugins. In case you do not change the permissions, the default permissions require administrator privileges that the CMake program does not have.

## 1.3. The QGIS OAuth2 plugin Development

The OAuth2 plugin for QGIS 2.18.x for Testbed 13 is based on the code from Boundless Geo (https://github.com/boundlessgeo/qgis-oauth-cxxplugin). During the Testbed 13 project, we have made changes to the sources and extended the functionalities to mainly support OAuth2 Dynamic Client Registration via Software Statements, as described in this ER but also to support manual finish of the Authorization Code Grant. The following is a basic list of changes and improvements:

*Table 8. Changes and improvements over the codebase from Boundless Geo*

| Change / Improvement | Description |
|---|---|
| state parameter | The Boundless Geo version of the plugin requests the state parameter to be provided by the user. We have changed that as we think that the user must not be responsible for providing that, as a duplication of a state parameter could lead to unintentional errors. The Testbed 13 version generates the state parameter automatically for each authorization request to the Authorization Server and checks the value from the redirect to ensure no CSRF attacks. |
| optional parameters | The Testbed 13 version populates all the extra parameters as provided by the user vi the GUI. This was important to support the setup from 52North using the Auth0 Authorization Server, because the optional parameter audience was required. |
| Dynamic Client Registration | The Testbed 13 version provides an additional configuration tab "software statement" which allows a user to automatically register the plugin with a required configuration with the Authorization Server. Of course this can only be leveraged, if the Authorization Server involved supports the registration via digitally signed software statements (JWTs) as described in this ER. |

| Change / Improvement | Description |
| --- | --- |
| Authorization Code manual | Support for the Authorization Code flow to copy and paste the authorization code when redirect_uri is not locahost or 127.0.0.1 |

# 1.4. The QGIS Plugin for OAuth2

Once the QGIS Plugin for OAuth2 is installed, you need to register it with the Authorization Server that is in charge of releasing access tokens for the Resource Server that is protecting the OGC Web Services that you like to protect.

This OAuth2 plugin is for general purpose use and not specific to a particular use with OAuth2. In this sense, you need to register the plugin with the Authorization Server specifying certain parameters that must be "negotiated" (can be obtained) from the entity operating the Authorization Server.

The plugin supports different methods for registration: (i) you can register the plugin manually with the Authorization Server and then use the manual configuration window to fill in the blanks; (ii) use a software statement released by the operator of the Resource Server that creates proper software features to access a particular service. It is entirely possible that the operator of the Resource Server will provide different access profiles resulting in different software statement. How to obtain these software statement, valid for you, is outside the scope of this documentation.

For making the entire registration more clear, we are going to do this based on the Authorization and Resource Server deployed by Secure Dimensions for demonstration purposes in the context of Testbed 13.

## 1.4.1. The Dutch Kadaster Resource Server

For the TIE, Dutch Kadaster operates an OAuth2 protected Resource Server that hosts a WMS for Othophotos of the Netherlands.

```
https://test.secure.geodata2.nationaalgeoregister.nl/lv-beeldmateriaal-poc/2015/wms
```

The associated OAuth2 Authorization Server is described in table no. 1 below.

## 1.4.2. The Plugin Registration Process

The QGIS Plugin for OAuth2 cannot be used for authorization without being configured for a particular use at a specific Authorization Server. Two options exist for how to register the plugin for a particular use at an Authorization Server: Manual or via Dynamic Client Registration.

**Manual registration**

The manual registration requires that the QGIS user visits the registration pages of the Authorization Server first. These pages differ in options and complexity, but the essential result is that the user obtains a "client_id" and a "clients_secret" as well as OAuth2 specific endpoints

required.

The following table represents the result of a manual registration using the Dutch Kadaster Authorization Server provided for Tested 13.

*Table 9. QGIS Plugin Registration at Dutch Kadaster OAuth2 Authorization Server*

| **Parameter** | **Value** |
|---|---|
| OAuth Grant type | ROPC |
| Token endpoint | https://authorization.test.kadaster.nl/auth/oauth/v2/token |
| client_id | a916f534-f22e-476e-af0e-3a3c692614b0 |
| client_secret | 7f947487-272b-415a-9d1b-e694d541da87 |
| Scope(s) | beeld |

In order to use these OAuth2 registration parameters, the user must also obtain a username and password. Please contact Frank Terpstra at f.terpstra@geonovum.nl [mailto:f.terpstra@geonovum.nl] to get an account.

#### Using Dynamic Client Registration and a Software Statement

The registration of the QGIS OAuth2 plugin via a software statement reduces the capabilities of the plugin (support for all OAuth2 grant types, scopes, etc.) to a specific use and thereby mimicking a Web-Application. Because the Dutch Kadaster OAuth2 Authorization Server for TB13 does not support Dynamic Client Registration, we are going to explain the procedure with the OAuth2 Authorization Server provided by Secure Dimensions: https://as.tb13.secure-dimensions.de

Bascially, the user that wants to connect to the OAuth2 protected Geoserver hosted at https://rs.tb13.secure-dimensions.de must use one of the available software statements. The software statement includes the required information to register the plugin with the affiliated Authorization Server.

After downloading the desired software statement (with your favorite Web-Browser) and stored on local disk, the user can open the Oauth2 plugin tab "Software statement" to use the file for registration. The user must load the software statement (.jwt) and optionally provide the ".well-known" URL for the Authorization Server: https://as.tb13.secure-dimensions.de/.well-known/openid-configuration This URL is used by the plugin to fetch required endpoints URL if the software statement does not contain the entry "register_endpoint". Which endpoints are required depend on the OAuth2 grant type expressed in the software statement. Common endpoints are the token and authorize endpoint.

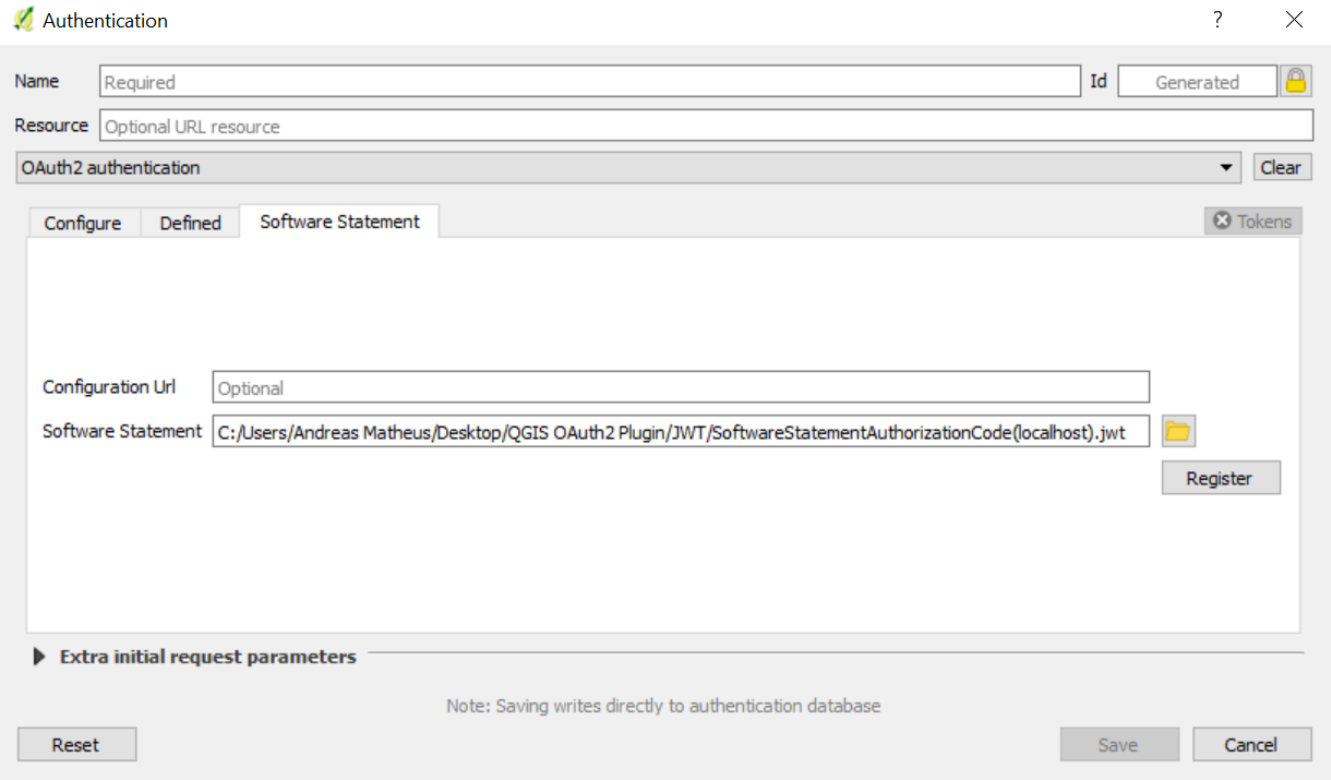


*Figure 57. OAuth2 Plugin for QGIS 2.18.4 showing register via Software Statement*

*Note*

```
For the configuration shown above, no Configuration URI is required as the
register_endpoint is comprised in the software statement.
```

Next, the user can register the plugin for a particular use as outlined in the Software Statement by click the "Register" button. After a successful response from the Authorization Server, the plugin populates the required values into the configuration tab and put focus on it.

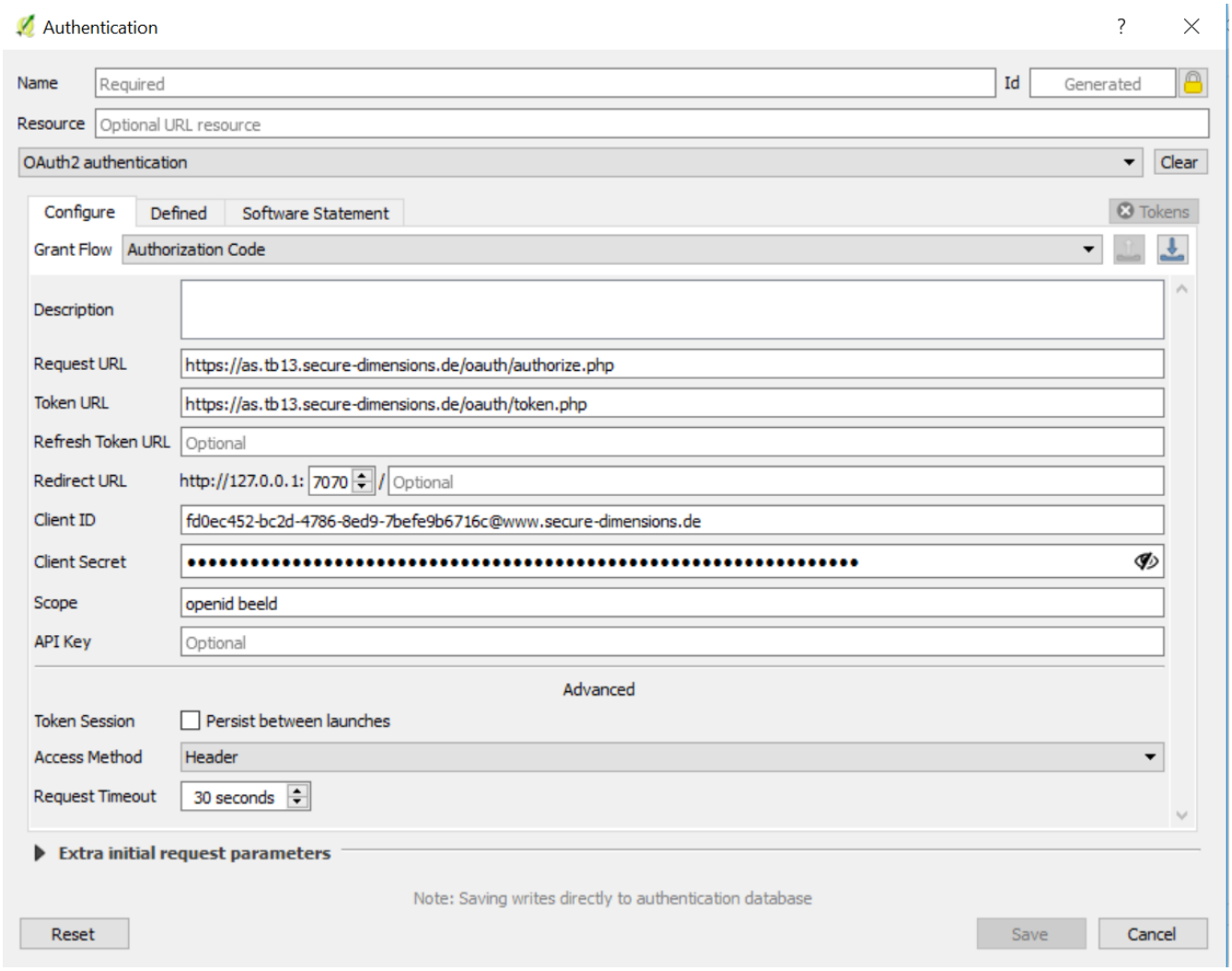


*Figure 58. OAuth2 Plugin for QGIS 2.18.4 showing software statement registration result*

The user can save this particular configuration by simply providing a configuration name. Then click the "Save" button on the bottom of the dialog.

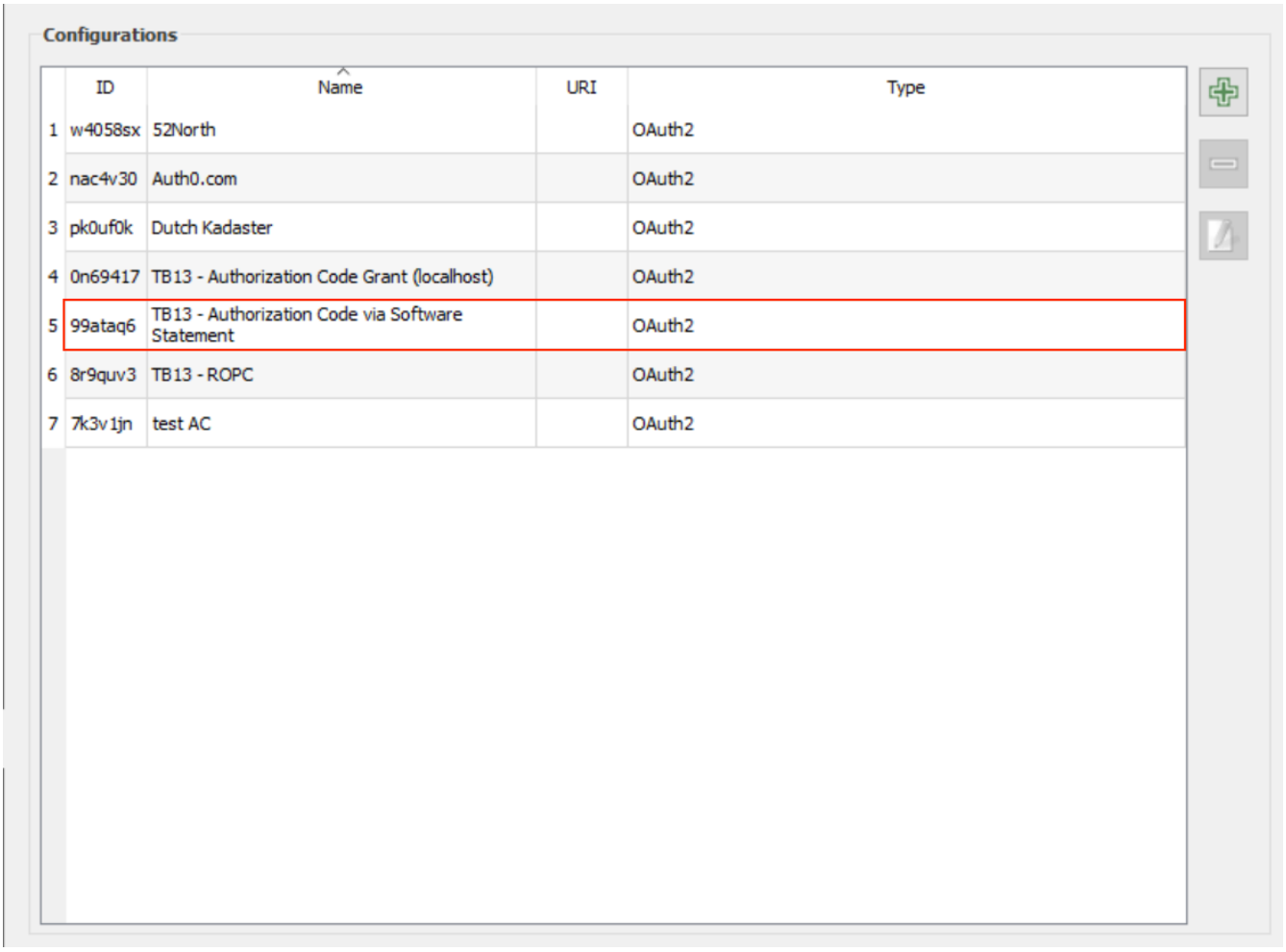


*Figure 59. OAuth2 Plugin for QGIS 2.18.4 showing software statement registration result*

The resulting configuration is highlighted in the figure Figure 59.

### 1.4.3. The Actual Use

Once the user has registered the plugin via a software statement that enables access to a particular resource (OGC Web Services in our case), the OGC Web Server configuration can be set. The user must simply add the OAuth2 configuration suitable to access the protected service.

As an example, the following configuration illustrates the linking of the just registered configuration to a WCS hosted at the Resource Server.

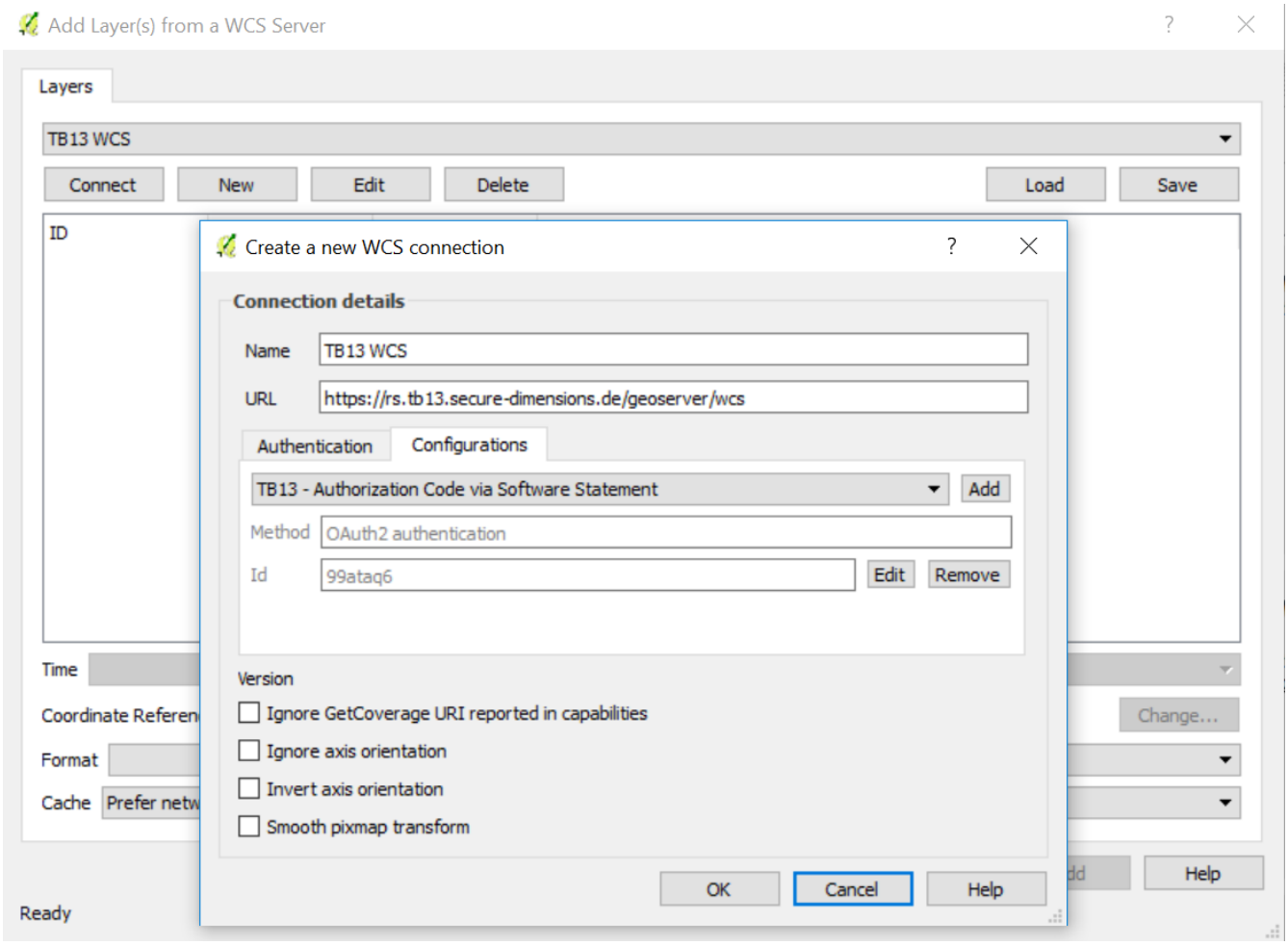


*Figure 60. OAuth2 Plugin for QGIS 2.18.4 used for a protected WCS*

# Appendix C: How to use the QGIS Plugin for SAML2

This section provides information on how to use the SAML2 plugin for QGIS.

## 1.1. Installation

The QGIS Plugin provided under OGC Testbed 13 is compiled as a Windows 32bit executable to work with QGIS 32bit build 2.18.4. However, the compiled DLL (saml22authmethod.dll) does also work with the QGIS 2.18.13 version.

The actual installation of the plugin is quite simple: You just download the plugin from this URL (https://sp.tb13.secure-dimensions.de/saml2authmethod.dll) and put the DLL into the QGIS "plugins" directory. This is usually located under the diretory for the QGIS installation.

*Note*

```
Assuming your QGIS installation is in C:\Program Files\QGIS-2.18.13 then you must copy
the QGIS plugin DLL (saml22authmethod.dll) into the directory C:\Program Files\QGIS-
2.18.13\apps\qgis\plugins
```

## 1.2. Compile from Sources

For Testbd 13, Secure Dimensions provides a compiled version of the plugin for easy installation (see previous section). But, it is also possible to compile from the sources. This procedure is more difficult (and only suggested for decent QGIS developers) as you need to build a couple of dependencies. It is not our intention to provide a step-by-step instructions list, but rather provide an overview so that you can go into the details yourself. For that purpose, we describe the build process based on QGIS version 2.18.4 (which was the latest available at the time of B13 start) and a Windows 10 32bit installation.

First, you need to install Microsoft Visual Studio 2010.

Second, build QGIS 2.18.14. How to do this, please follow the detailed description available from OSGeo: https://github.com/qgis/QGIS/blob/release-2_18/INSTALL

Third, obtain the sources for the QGIS SAML2 Plugin (master) from Github:

https://github.com/securedimensions/QGIS-SAML2-Plugin

Then, start cmake and use the plugin installation directory as source. Create a build directory and use that with cmake. Then select the compiler based on Visual Studio 2010 (use native compiler). The following figure illustrates the CMake configuration. Please make sure to uncheck the "build test app" option.

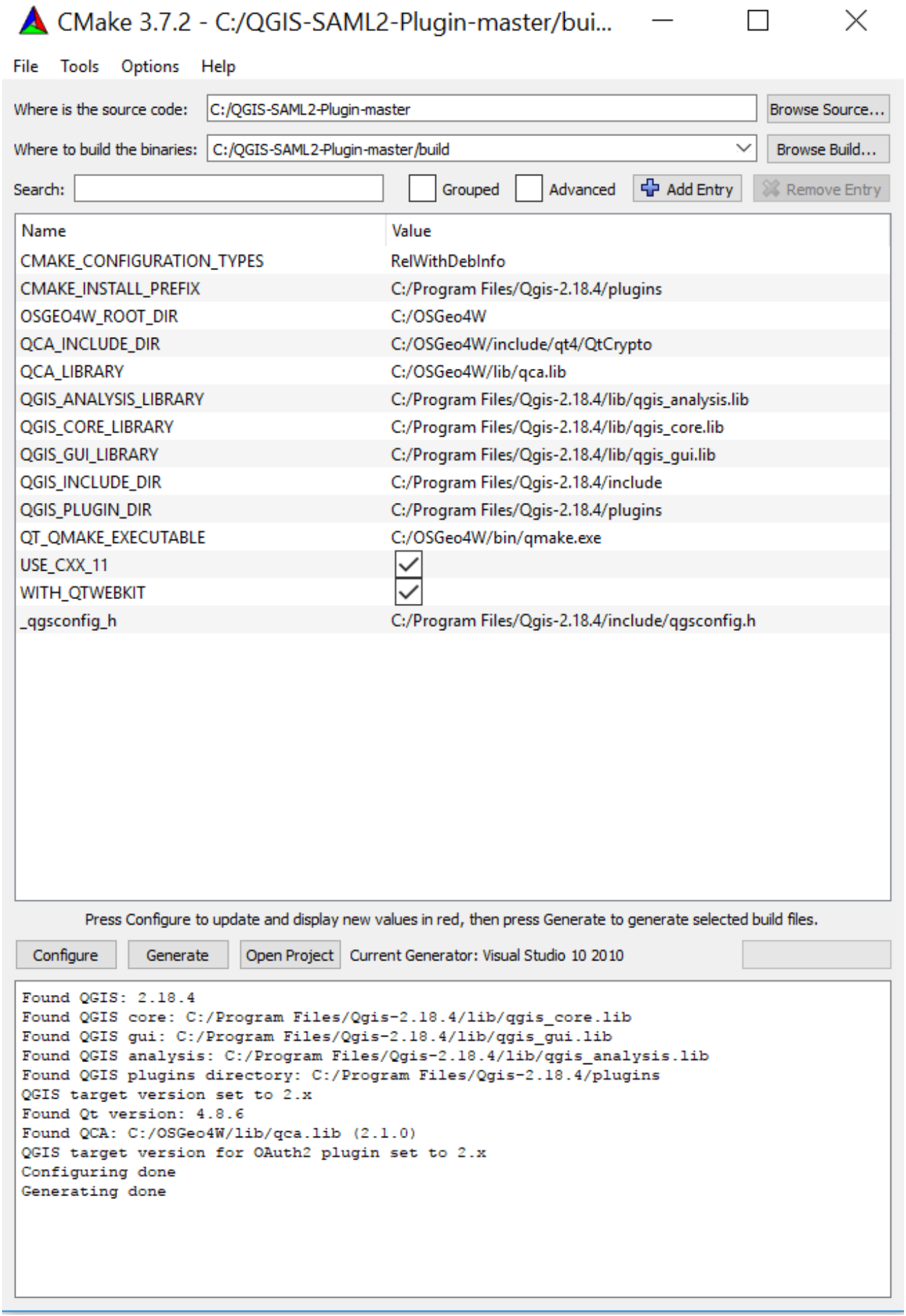


*Figure 61. CMake configuration for building the SAML22 Plugin for QGIS 2.18.4*

As can be seen, some dependencies exist to the OSGeo installation of QGIS (C:/OSGeo4W and C:/Program Files/Qgis-2.18.4/include/qgisconfig.h).

Final note: In order for the installation of the DLL to finish with no errors, it is important to change the permissions on the target directory to allow any access: C:/Program Files/Qgis-2.18.4/plugins. In case the permissions are not changed, the default permissions require administrator privileges that the CMake program does not have.

## 1.3. The QGIS Plugin for SAML2

The main difference of this SAML2 plugin compared to the OAuth2 plugin is that no registration is required. With SAML2 protected services, the plugin must execute the SAML2 ECP. For this protocol it is required that the user provides the IdP to use for login.

As noted from the SAML2 ECP standard, the IdP discovery is out of scope. This means that any implementation choice is acceptable as long it results in an IdP selection. For Testbed 13, we decided to implement a generic approach where the IdP discovery is based on the SAML2 federation metadata for the Service and Identity Provider. The Testbed federation metadata can be found here:

```
https://www.tb13.secure-dimensions.de/federation-metadata.xml
```

For a URL, provided by the user, the plugin parses the metadata and extracts all available IdPs and presents that list to the user. This simplified approach works best for a small number of IdPs (as currently in Testbed 13) but would not work quite as well with a large federation with many IdPs. Nevertheless, everyone has the freedom to implement their own IdP discovery as they see fit.

The following figure illustrates the simplified configuration of the plugin. For the ECP flow to work, the only input required are the user credentials and the federation metadta. When providing login credentials to the plugin, they will be saved in the encrypted SQL lite database as used by the QGIS authentication plugin. And, it is important to know that the user credentials are not send to the Service Provider!

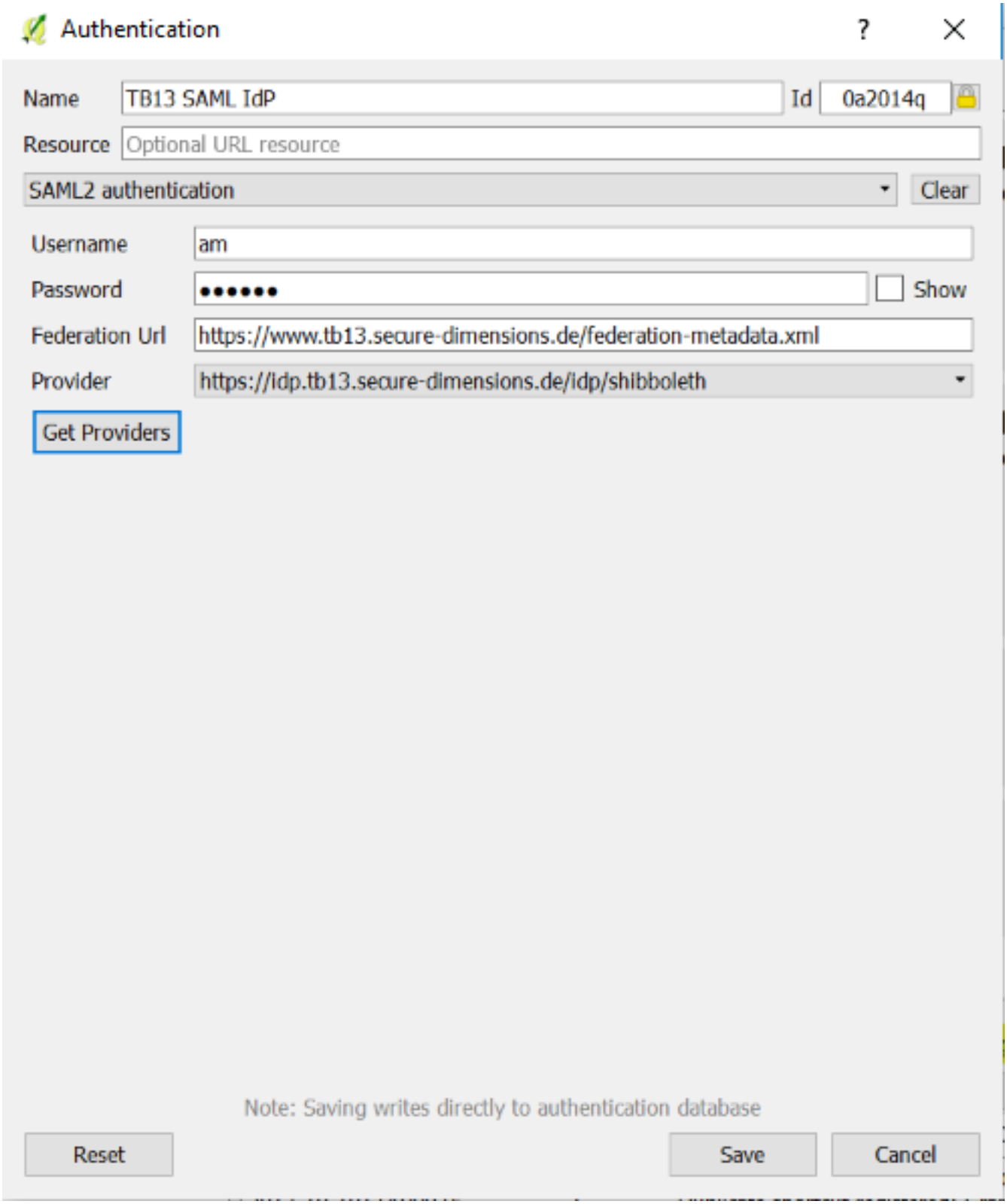


*Figure 62. SAML2 Plugin configuration example*

Once the SAML2 plugin is configured to work with a particular IdP, this configuration can be linked to any OWS, protected via SAML2 in the usual fashion. The following figure illustrates how to do that for a SAML2 protected WMS provided by Secure Dimensions for Testbed 13.

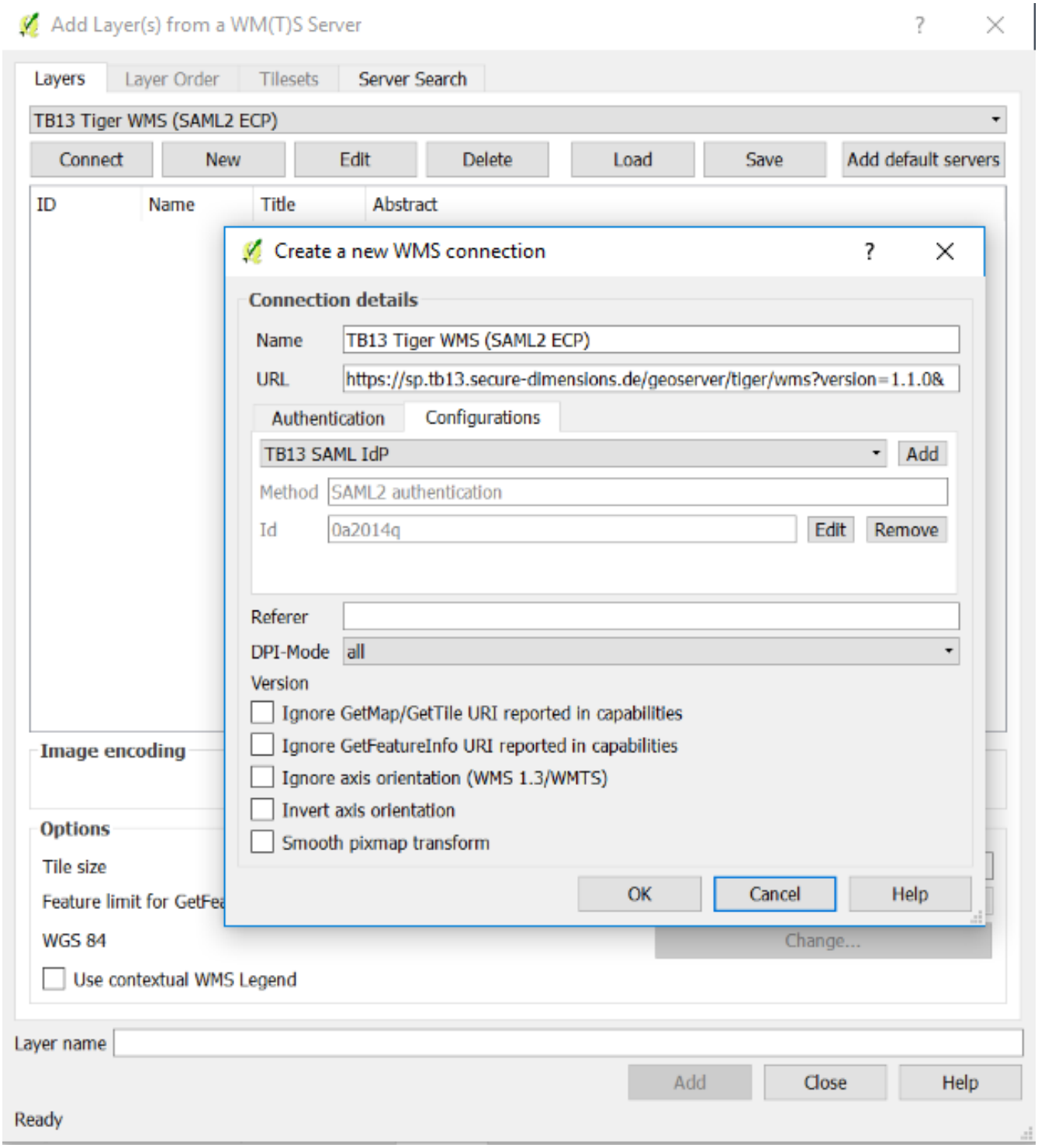


*Figure 63. SAML2 configuration used for SAML2 protected WMS*

# Appendix D: Revision History

*Table 10. Revision History*

| Date | Release | Editor | Primary clauses modified | Descriptions |
|---|---|---|---|---|
| June 30, 2017 | A. Matheus | .1 | all | PER version |
| September 26, 2017 | A. Matheus | .2 | all | DER version |
| October 06, 2017 | A. Matheus | .3 | all | DER version update |
| October 13, 2017 | A. Matheus | .3 | all | Preperation for Pending Documents |
| October 23, 2017 | A. Matheus | .4 | all | Final edits |
| November 07, 2017 | A. Matheus | .5 | all | Spell checking |

# Appendix E: Bibliography

None

[1] Open Geospatial Consortium, Inc, 2012, Public Engineering Report for the OWS Shibboleth Interoperability Experiment, https://portal.opengeospatial.org/files/?artifact_id=47852