OGC Testbed-14

Security Engineering Report

Table of Contents

Publication Date: 2019-03-05

Approval Date: 2018-12-13

Submission Date: 2018-10-29

Reference number of this document: OGC 18-026r1

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D024

Category: Public Engineering Report

Editor: Juan José Doval, Héctor Rodríguez

Title: OGC Testbed-14: Security Engineering Report

OGC Engineering Report

COPYRIGHT

Copyright (c) 2019 Open Geospatial Consortium. To obtain additional rights of use, visit http://www.opengeospatial.org/

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

This Security Engineering Report (ER) covers several OGC Testbed-14 topics:

- Best practices for the integration of OAuth2.0/OpenID Connect services
- Mediation services for different security environments
- Federated identity management
- Securitization of workflows

The first two topics are the main focus of this ER. During this Testbed, a server that provides OAuth2.0 and OpenID Connect capabilities was extended with a mediation service that allows for a centralized security authority with users/clients that implement different security standards.

The remaining two topics will expand on the close relationship between Security, Workflows and Federated Clouds and the respective implementation challenges. On these specific topics, this ER also outlines and provides a proof-of-concept for a simplistic architecture approach that explores one of several Federated Clouds architectures.

1.1. Requirements & Research Motivation

The motivation behind this ER resides on the prevalent disconnect between security standards and current OGC standards. It is necessary to analyze the needs of Spatial Data Infrastructure (SDI) regarding securitization, taking also into account how it affects usability and interoperability.

This ER acknowledges the complexity of this task and focuses on performing a complete analysis of these needs (taking into account workflow and federation aspects of security) and lays out the groundwork for a complex architecture that fulfills as many of those needs as possible.

Therefore, the requirements addressed by this ER are to:

- Perform an initial state-of-the-art analysis of the current security standards and the security needs of SDI regarding Authentication, workflow execution and federated identity management.
- Propose a simplistic view of a security architecture that serves as an initial step towards more complex solutions that fulfill all security requirements. Outline all future steps as precisely as possible.
- Implement an Authentication and Authorization server based on OAuth2.0 and OpenID Connect that is also able to interact with other authentication methods (besides tokens) such as X.509 certificates (used in Transport Layer Security authentication or TLS which is the basis for HTTPS) and SAML-based security environments.
- Analyze and include Workflows and Federated Clouds requirements that have an impact on the resulting architecture.

1.2. Prior-After Comparison

Security discussions prior to Testbed 14 were often focused on specific solutions that solved only a handful of security requirements and were limited to custom-built systems.

On the integration of OGC standards and definition of best practices for integration, the OWS Common Security Standards Working Group recently issued an OGC Web Services Security Standard candidate (OGC 17-007r1) that aims to standardize the security aspects of current Web Services Standards while providing backward compatibility and interoperability. On 13th August 2018, the OGC Technical Committee (TC) voted to recommend approval of the OGC Web Services Security specification as a new OGC standard. The OGC® Web Services Security standard was published on the 28th January 2019.

On the other hand, security discussions prior to this Testbed included concerns regarding Federated Identity Management, but these were not specifically addressed in enough detail.

This situation highlights the need for a general approach to security architectures that utilize the current state-of-the-art standards and takes into account Security Standard Candidates while providing a set of best practices on the integration of security on Federated Clouds and Workflows.

As a result of activities performed during this Testbed, the analysis and implementations documented on this Engineering Report serve as an initial step towards a complete high-level architecture that addresses the identified limitations of current standards when applied to SDI.

Using this ER as a reference, a more general solution to security requirements is proposed (including workflow and federated clouds concerns), making it easier to further discuss and implement complex security architectures with clearly identified components that any Security Environment (in the context of Access Management) or Administrative Domain (in the context of Federated Identity Management) should have.

1.3. Recommendations for Future Work

NOTE

Summary

This subsection acts as a summary for the contents of all future work sections throughout this document.

As a result of the work performed during this Testbed, several future work points have been identified:

- This Testbed has resulted in a broad authorization scheme, provided by the Authorization Server, that allows authorization based only on user identity. In order to extend on these capabilities, fine-grained authorization methodologies need to be further explored. Following the current direction of using OAuth2.0 based technologies, the **UMA standard** (User Managed Access)[1] would allow authorization based on OAuth2.0 scopes and policies previously defined by the service owners.
- It has become clear that OAuth2.0 and OpenID Connect are heavily focused on interacting with end-users and, while this enhances user experience (even more when coupled with Single Sign-On), SDI still requires automation of processes during the execution of workflows (Mediation

and Delegation capabilities). To solve this, **it is necessary to further analyze the usage of the Client Credentials and Resource Owner grants** for their application on workflows.

- The automation of process execution mentioned on the previous point could benefit from the usage of JavaScript Object Notation (JSON) Web Tokens to store signed and encrypted user information. It is highly recommended to **extend the developed deliverables to be able to handle JSON Web Tokens** efficiently.
- During the design and development of the Mediation Service for this Testbed, it has become clear that the current architecture lacks **a component that should be able to utilize OpenAPI and capabilities documents, combined with a catalog of service** references to completely automate the acquisition of user tokens and their use on workflows (avoiding unintended leaks of user information as explained on the Workflows section of this ER).
- One of the main issues raised on the Workflows components mainly concerns the usage of secured services outputs as unsecured service inputs. This needs to be solved by either extending the constraints on discovery documents (or OpenAPI documents) or extending capabilities of these services as explained on 8.2
- The last remaining component that the resulting architecture lacks is a **Federation Manager capable of allowing management of resources** on the owner side that should facilitate the solution of governance issues raised on the Federated Clouds ER of this Testbed.
- During the design and development phase of the security architecture, the Federated Clouds ER discussion has raised relevant points on the topic of federations between Administrative Domains based on different architectures. This means that federation with SAML entities such as eduGain is still a relevant topic that should be addressed on future Testbeds.

1.4. Document contributor contact points

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

Contacts

Name	Organization	Role
Héctor Rodríguez	Deimos Space	Security ER Editor
Juan José Doval	Deimos Space	Security ER Editor

1.5. 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.

- OGC: OGC 17-007r1, OGC® Web Services Security Standard, http://docs.opengeospatial.org/is/17-007r1/17-007r1.html
- OGC: OGC 06-121r9, OGC® Web Services Common Standard, https://portal.opengeospatial.org/files/?artifact_id=38867&version=2
- 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 7662 - OAuth 2.0 Token Introspection, https://tools.ietf.org/html/rfc7662
- IETF: RFC 7662 - OAuth 2.0 Dynamic Client Registration Protocol, https://tools.ietf.org/html/rfc7591
- OIDF: OpenID Connect Core 1.0, http://openid.net/specs/openid-connect-core-1_0.html
- IETF: RFC 7519 - JSON Web Token (JWT), https://tools.ietf.org/html/rfc7519

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. Definitions

- Security Environment

  ```
  A system that securely manages end-user information for the purpose of providing
  Identity Management and Access Control.  These capabilities are usually achieved by
  using cryptographic methods, secure network design, and observing data protection
  regulations.
  ```

- Administrative Domain

  ```
  A system that augments a Security Environment by defining a desired set of policies
  and governance for managing users and resources.  This includes defining an
  authoritative source for policy concerning resource discovery and use.
  ```

- Federated Environment

  ```
  The creation and management of a Virtual Administrative Domain whereby the same
  kind of policies and governance can be used to manage users and resources within
  the VAD that are, in fact, coming from an arbitrary number of non-federated
  Administrative Domains.  This depends on a Federated Security Environment.
  ```

3.2. Abbreviated terms

- BPMN Business Process Model and Notation
- IDP Identity Provider
- NextGEOSS Next-generation Global Earth Observation System of Systems
- OIDC OpenID Connect
- SAML Security Assertion Markup Language
- UMA User Managed Access
- WFS Web Feature Service

Chapter 4. Overview

Section 5 contains a brief state-of-the-art analysis on the security topic, covering and comparing all currently used technologies and how they fulfill security requirements.

Section 6 analyzes the Authorization Server designed during this Testbed, how it conforms with the current standards, the methods that are made available for authentication and the integration efforts made on clients and services (such as Web Feature Service -WFS- 3.0).

Section 7 discusses the mediation capabilities exercised during this Testbed and the effect these have on allowing access from other security environments. It provides an analysis on more advanced strategies that could solve industry needs.

Section 8 presents the solutions implemented to securitize Workflows using different authentication flows. It includes recommendations on how to adapt current Workflow and chaining solutions (such as Business Process Model and Notation, BPMN) in order to adhere to security best-practices and recent OGC Security standards.

Section 9 provides a description of the federation efforts with external entities (such as NextGEOSS). The resulting architecture is analyzed only from a technical point of view as a recommendation of possible technology candidates that the Federated Clouds task could take into account on their separate analysis.

Section 10 summarizes all security related Technology Integration Experiments (TIEs) performed during this Testbed and references detailed results on their respective Engineering Reports

Annex A provides an integration guide that explains the main Endpoints of Identity Providers based on OpenID Connect. It is mainly meant for developers.

Chapter 5. Security State of the Art

The main objective of security when it comes to Spatial Data Infrastructure is to provide a Triple-A architecture: Authentication, Authorization and Accounting. The first two help on adding more security layers on top of the usual ones (such as firewall usage, and organizational security policies), while the last allows traceability of the actions of the users while also enabling billing and quoting operations.

This section aims at providing context to the reader regarding Security, application level standards and the methods for authentication and authorization that they provide, identifying which of these are being exercised during this Testbed but also mentioning prospective work regarding accounting operations.

5.1. Security standards

There are four main standards that need to be taken into account:

- **OAuth 2.0**: It is a standard protocol for authorization. OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006, with October 2012 as its ratification date. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.
- **SAML 2.0**: Security Assertion Markup Language 2.0 (SAML 2.0) is a version of the SAML standard for exchanging authentication and authorization data between security domains. SAML 2.0 is an XML-based protocol that uses security tokens containing assertions to pass information about a principal (usually an end user) between a SAML authority, named an Identity Provider, and a SAML consumer, named a Service Provider. SAML 2.0 enables web-based authentication and authorization scenarios including cross-domain single sign-on (SSO), which helps reduce the administrative overhead of distributing multiple authentication tokens to the user. SAML 2.0 was ratified as an OASIS Standard in March 2005, replacing SAML 1.1.
- **OpenID Connect (OIDC)**: It is an extension to OAuth2.0 protocol that adds a simple identity layer on top the already defined specifications. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner. OpenID Connect allows clients of all types, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. The specification suite is extensible, allowing participants to use optional features such as encryption of identity data, discovery of OpenID Providers, and session management, when it makes sense for them.
- **User-Managed Access (UMA)**: It is an OAuth-based protocol designed as an extension to OAuth2.0 standard protocol that gives a web user a unified control point for authorizing who and what can get access to their online digital data (such as identity attributes), content (such as photos), and services (such as viewing and creating status updates), no matter where all those things live on the web.

Several gaps can be identified on current Authentication, Authorization and Accounting services such as those made available to science users and application developers on exploitation platforms. Nowadays, most of the security environments use the SAML standard for solving the authentication

and Single Sign-On requirements.

Some of the security environments use XACML (eXtensible Access Control Mark-up Language) for authorization, managing it at application/service level but require a lot more effort in order to have centralized trace of user access that allows accounting. This effort is also passed to developers of client applications which usually are more used to Authentication and Authorization schemes that tend to be JSON-based.

Most of the security environments of the industry consist of either a single point of entry to a restricted system that allows access to all services without any kind of discrimination, or a conglomerate of services with different authentication and authorization schemes. Both of these solutions have an impact on the usability of the system, as well as their ability to scale up easily.

Some of these environments are already making use of identity providers by means of SAML, solving the Single Sign-On requirements that smooth the usability issues on the user side, but fail to address more complex technical requirements such as the use of distributed and aggregated user claims, session timeouts, gathering of user consent to share their personal data and the most important of them: the ability to extend the standard to provide new functionality.

Protocols such as OpenID Connect and UMA 2.0 can be used in combination with or aggregated to OAuth2.0 to provide the means of solving more complex technical needs without extensive development effort. OpenID Connect acts as the counterpart for SAML, and can be combined with UMA in order to fulfill all the requirements of Accounting for current services.

Another key aspect is that development and integration of new systems will not be as complex as the systems previously mentioned thanks, for example, to the Representational State Transfer (REST) Application Programming Interfaces (APIs) that will be made available to all the components in the infrastructure. These components will be able to make use of these APIs to leverage authorization without user interaction, also allowing secure interactions between components.

During this Testbed, OpenID Connect and OAuth2.0 were used taking into account that there is a need to reduce the development and integration impact on applications, services and users while also allowing centralized authentication and authorization with Single Sign-On.

5.2. Authentication methods

NOTE	This subsection does not contain technical detailed explanations on authentication methods. Developers and software architects can find this information on Annex I

When using OpenID Connect, the main advantage over OAuth2.0 is that the client has the possibility of acquiring user information from the Authorization Server (and this can enable federation as shown in section 9). Additionally, several ways of authenticating users and clients are made available with Each of them having a specific purpose and a best use scenario.

Knowing what these scenarios are in advance can be useful to find a compromise between usability and security. It is also important to take into account the main weaknesses and strengths that each of these methods imply, and adjust the security architecture accordingly. There are four main credential grants that make use of the following endpoints:

- **Authorization Endpoint**: This endpoint triggers a redirection by default to the main portal of the Authorization Server, and after authenticating the user, redirects back to a callback Uniform Resource Locator (URL) including a code (or a token in case of Implicit Grant flow) that can be used to advance further on the grant procedure.
- **Token Endpoint**: This endpoint releases a token when receiving the appropriate parameters (i.e. a code in case of Authorization Code flow).
- **User Info Endpoint**: This endpoint allows extraction of user information when providing an access token, enabling authorization and accounting.

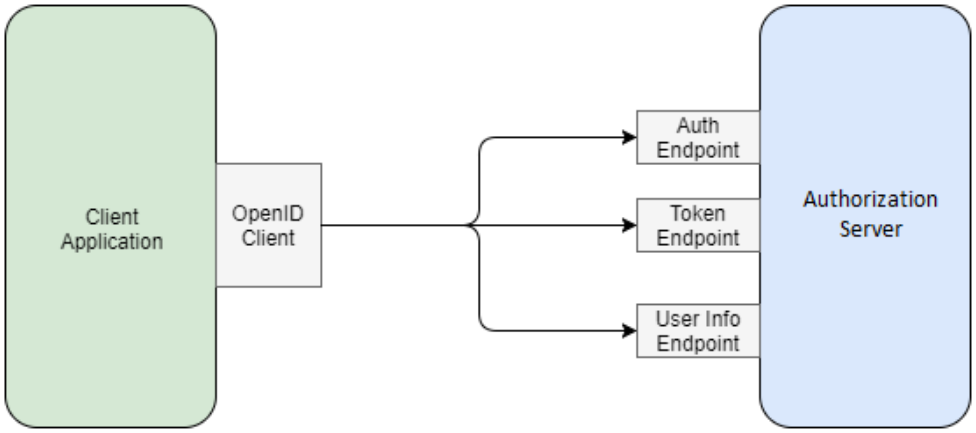

Figure 1. Endpoints made available by OpenID Connect

Available grants are:

- **Implicit Grant**: The main feature of this grant is that it forfeits client authentication in exchange of an easier implementation and better user experience and this makes it a "weak" solution security wise. Both clients developed during this Testbed (explained on section 7.2.1) are compliant with this flow, and workflows (section 8) will participate on the grant by relaying tokens within their operations.

NOTE

Recommended uses of this grant

This grant is mainly used on Single Page Applications [2] and clients that do not have any back-end development and it also implies that the user information that is made available through this grant should disclose as little information as possible.

- **Authorization Code Grant**: This grant requires an additional step before getting access tokens compared to the Implicit Grant. Whereas the previous grant acquires tokens by just passing credentials to the Authentication Endpoint, in this case the Authentication Endpoint will just deliver a code, that then can be exchanged for a token if the client authenticates itself. For this reason, this is the strongest grant in terms of security, and it has been exercised during this Testbed by one of the clients (as explained in section 7.2.1)

- **Resource Owner grant**. This grant requires the user to trust the client completely. The user needs to provide credentials to the client and the validation is performed by said client. This previous step of transfer of credentials can be performed in advance, making the process completely automated, with no user interaction.

NOTE

Recommended uses of this grant

This grant is identified on section 7 as a very promising solution for Mediation Services that try to implement other standards as a layer on top of OAuth2.0 (such as Web Processing Service (WPS) or Web Coverage Processing Service (WCPS). It is also useful for validating the functionality of an Authorization Server (section 6.2) without having to develop whole interactions like the previously mentioned grants.

- **Client credentials grant**. In this case, the end-user is not being authenticated. No user credentials are needed, because this flow aims only at authenticating the client itself. This grant is useful when there is no end-user, or in situations where the creation of a "default" user seems the only way to enable authentication of a process.

NOTE

An example of use

For example, if a WPS process were to be automatically triggered with no user interaction, but needed to access a Catalogue Service for the Web (CSW) which was protected behind an access token requirement.

Normally, user interaction could be required, but this grant allows the WPS to access a CSW without user interaction. This has huge implications when a service is driven by more than users (event-driven, time-driven or data-driven architectures). The Mediation Service utilizes this, as explained on section 7.

5.3. Authorization methods

It is known that OAuth2.0 does not really provide authentication capabilities [3] and that can be solved using OpenID Connect (which proves the identity of a user). But more interestingly, OAuth2.0 does not provide Authorization itself as well, but just the means of acquiring a set of access rights with no policies or rules attached. This functionality gap needs to be solved by either extending functionality on the client side, or implementing an extension of OAuth that enables authorization, and choosing one of these two solutions depends on the use cases.

5.3.1. Authorization based on scopes

It is possible to define a list of attributes indicating which users can or cannot access a service/resource. These attributes can be associated to an OpenID scope that allows applications to discriminate the access to the resources/operations between users.

All this information can be used to provide claims about users interacting with external clients and using the authentication flows previously mentioned.

Since this is the approach chosen for the Testbed, it is explained here in detail:

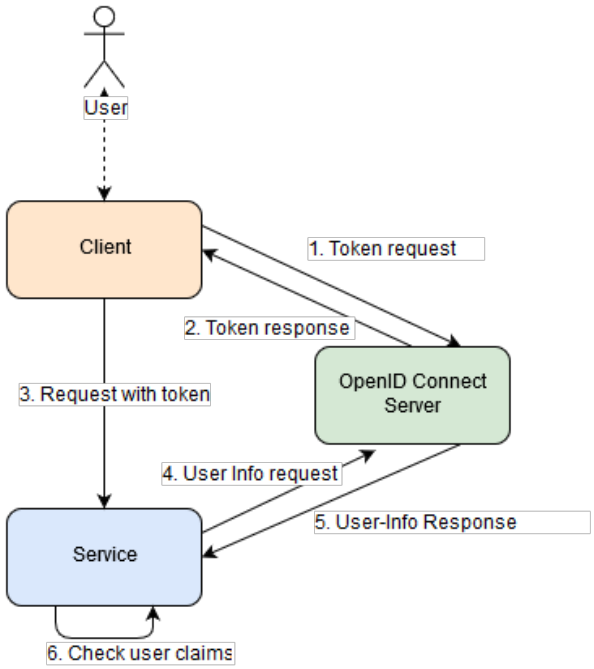

Figure 2. Authorization based on scopes

Server-Side Implementation

Any component within the security environment should define which user attributes are required to interact with its services. After that, any received request must contain an access token (obtained using the Authorization/Token Endpoint). This access token can be included in an internal request to the Authorization Server User Info Endpoint, which returns the set of default attributes that can be checked internally by the service in order to determine if access should be granted.

Client-Side Implementation

Any client trying to connect to a service must request an access token in order to provide claims about the user. This can be achieved with no user interaction by calling the Token Endpoint directly, using Resource Owner Grant.

5.3.2. Authorization using UMA

This authorization scheme can be considered an upgrade over authorization based on scopes. The main difference is that access policies and constraints are applied on a centralized authority which is usually close to the Authentication and Authorization Server in order to consume information regarding available scopes and attributes. Using this scheme, services can register and protect their resources assigning a set of required user claims that need to be fulfilled in order to access the resource.

These claims can be based on different kinds of constraints, like time of access, geographical situation of the resource or origin of the request, a set of resources that could be depleted (such as processing hours) or any other constraint based on user attributes or identity. **This enables smart accounting, billing and quoting, and could be subject of future Testbed work**.

5.4. Encryption

Security is not only about authenticating, authorizing users and having a trace of every action a user does. The security of communication channels must always be taken into account as well, especially when performing these procedures of authentication. This raises a valid concern when releasing tokens that could leak user information if intercepted by someone else.

In this case, OAuth2.0 (which is a layer of implementation behind OpenID Connect) manages these security concerns by relying on TLS (also called HTTPS/SSL) which is universally implemented on clients, servers and web browsers. This automatically requires that any endpoints made available through an Authorization Server need to be locked on HTTPS.

Additionally, releasing user information can be further securitized with OpenID Connect standard JSON Web Token (JWT) data structures. These can be used when claims that belong to a user need to be signed, generating signed JSON Web Tokens (JWS) and when their contents need to be obfuscated, in that case using encrypted JSON Web Tokens (JWE). Signing and encrypting this information with a public key and forcing the receiver to use a private key (similar to TLS foundational architecture) avoids interception of sensible information.

Using this JSON Web Token approach makes OpenID Connect dramatically easier for developers to implement, and in practice has resulted in much better interoperability, while also defining a JSON schema that avoids malicious or unauthorized data transfers [4].

Chapter 6. Authorization Server

6.1. Design Overview

Given that the Authorization Server needs to be compliant with OAuth2.0 and OpenID Connect (References), there is not much design effort to be made regarding its API endpoints, but it might be interesting to take into account further needs (such as federation) and interactions.

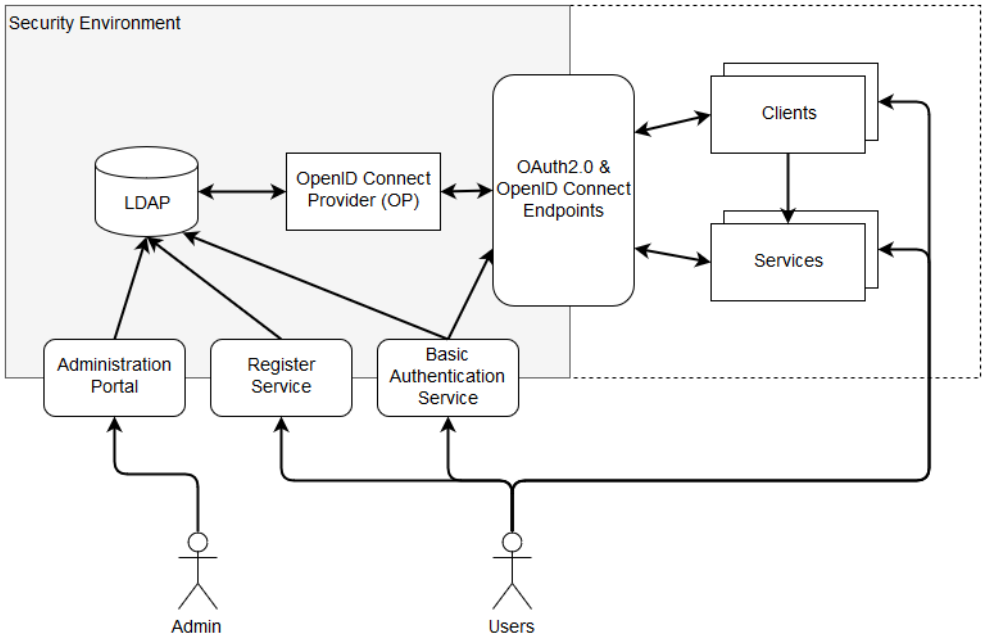

Figure 3. Structure of an OpenID Connect security environment.

In essence, the authorization server adds registration and management capabilities (that populate the LDAP service) on top of the authorization endpoints and makes information about the service available by means of an OpenID Connect Discovery document that lies on a commonly used relative path (/.well-known/openid-configuration).

The Basic Authentication Service is made available for potential use on profile management, and can either use LDAP credentials or utilize the authentication and authorization scheme. The end-users can then interact with clients to retrieve tokens and access services.

Although all grants (section 5.2) are made available using this Authorization Server, only the Implicit Grant will be used to provide authentication and authorization capabilities to workflows. This is due to the fact that the client implementation that will interact with workflows is purely based on JavaScript, which does not allow one to effectively authenticate the client in addition to the end-user (client credentials would be readable even if obfuscated).

Using the Implicit Flow will result in a token being passed between Testbed components. This token effectively contains user information since it can be used on the User Info Endpoint (section 5.2), making it a potential security flaw even if the communications between components are performed on a private network. In order to solve this issue, it is necessary to define the implementation of a different OAuth2.0 grant and the inclusion of encryption as future work.

6.2. Interface Context

During Testbed-14, several deliverables require interaction with the service described in this section (D151 Authorization Server). These deliverables can be either clients or services:

- Clients:
  - D142 - Next Generation Client Implementation 1 (GISFCU)
  - D143 - Next Generation Client Implementation 2 (Solenix)
- Services:
  - D140 - Next Generation Service Implementation 1 (WFS3.0 instance provided by Interactive Instruments)
  - D113 - Next Generation Service Implementation 2 (Two WFS3.0 instances provided by GeoSolutions and CubeWerx)
  - D146 - NGA Analysis Tool WPS (52 North)

Each of these implementations benefit from the Authorization Server by either acquiring tokens using one of the available grants, or validating tokens and extracting user information. The main flow is depicted in Figure 4 as an example for a Web Feature Service:

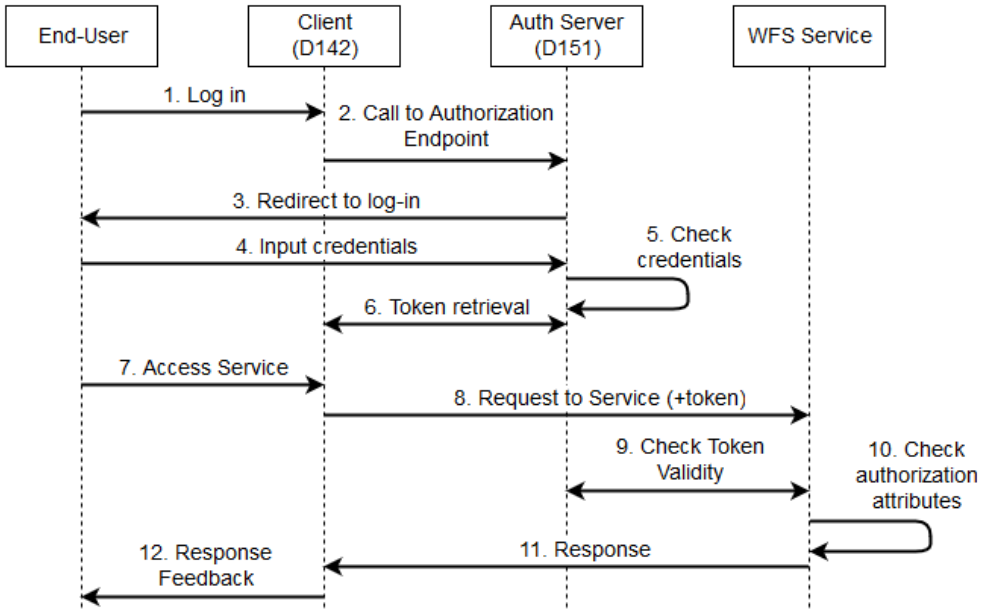

Figure 4. Base sequence diagram for the interactions that an Authorization Server enables.

This figure shows two crucial points for:

- Authentication (step 5). If the credentials that the user introduces on the Authorization Server are not valid, the flow would stop there since it is not possible to retrieve a token.
- Authorization (steps 9 and 10). If the authorization attribute checked by the Service (i.e. user_name) does not match the expected value or if the token is not valid (on step 9) the response should be an HTTP Unauthorized error.

6.2.1. Client-side approach

Any client application that wants to generate tokens on behalf of end-users is required to be registered as a client at the Authorization Server. The default behavior for this type of platforms is to require interaction between administrators of both systems that need to exchange a series of parameters (described in the Integration Guide in Annex I). This methodology is usually enough for most Security Environments, but the purpose of this Testbed-14 task is to go even further into the discovery of security requirements and creation of dynamic clients.

Dynamic client registration is a service specified in RFC 7591 that allows a client to generate profiles on the Authorization Server that have a limited validity period. Once this client has registered itself, it can be used by end-users to authenticate and acquire tokens. In the Testbed-14 scenario, the Next Generation Client implemented in Deliverable 142 serves as a completely integrated example of this functionality, that can be easily reproduced following the steps on A.2.2

All main grants, with the exception of the Client Credentials grant, have been leveraged by clients of this Testbed. The workflow components (assisted by D143) utilize tokens acquired via the Implicit Flow, while the rest of Next Generation Services will be receiving tokens acquired through D142 using the Authorization Code, Resource Owner Credentials and Implicit Flow grants. Additionally, the Mediation Service also acts as a client of the Authorization Server authenticating itself on a hand-shake operation based on the Client Credentials grant.

6.2.2. Service-side approach

As mentioned in section 5.3.1, the services will be providing authorization capabilities based on the user information tied to the received tokens. Authorization can be performed based on an attribute associated with the profile of the end-user or the service can choose to remain completely open and just rely on simple authentication of users.

The behavior shown by services is to expect a token encoded as Bearer Token on an Authorization header of the request. Extraction then becomes fairly simple, allowing for an easy integration of a secured process.

6.3. Analysis and Future Work

The result generated by this infrastructure, applied directly to a client-service architecture, is a simple and fairly secured access to service based on user authentication.

Implementation on the side of the service requires less implementation effort to be functional, but should also take into account the advertising of the service as secure. This aspect of secured web services is depicted in the OGC Web Services Security Standard that recommends the use of additional constraints to facilitate the discoverability of security requisites imposed on OGC Web Services [5] that are expected to be deployed over HTTPS.

This problematic association with the discoverability of service capabilities is also covered in D021 - Next Generation Web APIs/WFS 3.0 ER (OGC 18-045), that also delves deep into OpenAPI capabilities in this regard. During this Testbed, WFS 3.0 services had to understand the requirements and steps for implementing OAuth2.0 and OpenID Connect workflows and how to publish these requirements through OpenAPI documentations which became an addition to the current development status of

the services themselves.

This also raised a specific need for web services that provide HTML support and would also require the definition of dedicated clients on the Authorization Service, which means that some WFS 3.0 services developed during this program eventually had to identify themselves not only as services but also as "front-end" clients.

On the other hand, clients that consume these services had no unified mechanism to access protected resources and this particular situation forced them to understand the security criteria required for each service provider prior to their interactions. During this Testbed, the clients' main goal was to traverse the learning curve for OAuth2.0 and OpenID Connect leveraging of APIs, given that one of the first identified needs was to be independent of third-party implementations that could be harder to maintain. The interaction between the Authorization Server and client application developers has resulted in an extensive integration guide (Annex I) that aims to ease the learning curve.

Finally, when it comes to the resulting Security Environment enabled by the Authorization Server, the adoption of OpenID Connect has been a key factor that enables the automation of the discovery steps that need to be performed on the client side. The ability to parse discovery documents, coupled with the use of OpenAPI and constraints on the describe documents of OGC services allows clients to understand which are the security requirements for a given service and to dynamically generate clients that are fit for use.

The main points of identified future work for Authorization Servers reside on these topics:

- Further extending security on a Web Service level: this requires the usage of JSON Web Tokens in exchange for simple bearer tokens provided by OAuth2.0. OpenID Connect already gives support to this type of tokens, and this would allow signing and encryption of potential sources of user information, which would add on HTTPS/TLS connections to prevent Man-In-The-Middle attacks while avoiding an increase of complexity.
- Adoption of additional OAuth2.0 grants enabling a wider variety of methods to consume services beyond end-user driven executions. For example, the adoption of the Resource Owner or Client Credentials grants would allow the execution of processes triggered by events that are generated by an architecture itself (data driven or time driven).

Chapter 7. Mediation Service

7.1. Design Overview

The mediation service needs to provide the means of authentication to users that belong to different security environments, enabling federation and populating the internal user database with the information that these external environments grant access to.

As an ideal scenario, a mediation service should support the following standards:

- SAML: the main reason behind the inclusion of this standard would be exclusively the fact that the space industry data and service consumers still operate using SAML-based federations. As explained on section 5, OpenID Connect provides clear advantages and this means that this capability of the mediation service should only exist for the purpose of backwards compatibility.
- OpenID Connect: this is the standard that this Testbed focuses on, given that it facilitates the integration of Social Login and federation without having to forfeit usability in exchange.
- Mutual authentication (X.509 certificates): this comes as a requirement for organizations that are able to act as a Certificate Authority and usually provide personal certificates to members (this can be present in the form of electronic cards).

This Testbed focuses mainly on OpenID Connect and mutual authentication needs, leaving SAML integration out. In any case, the Mediation Service can only enable "inbound" authentication. This means that what this service accomplishes is mainly allowing access to users coming from other security environments (under predefined restrictions) and the rational behind this is explained in section 9. This service is unable to grant "outbound" access, but it is implied that external identity providers could also implement the same solution, allowing what is known as "two-way federation".

The preliminary analysis for this functionality provides several solutions, each of them raising different issues.

7.1.1. Proxy solutions

Coming up with proxy solutions is a very common situation when facing architectural problems with interoperability as the root cause. It is usually the solution with the least impact over current solutions and it basically overrides the need for standardization. So, why shouldn’t it be the case?

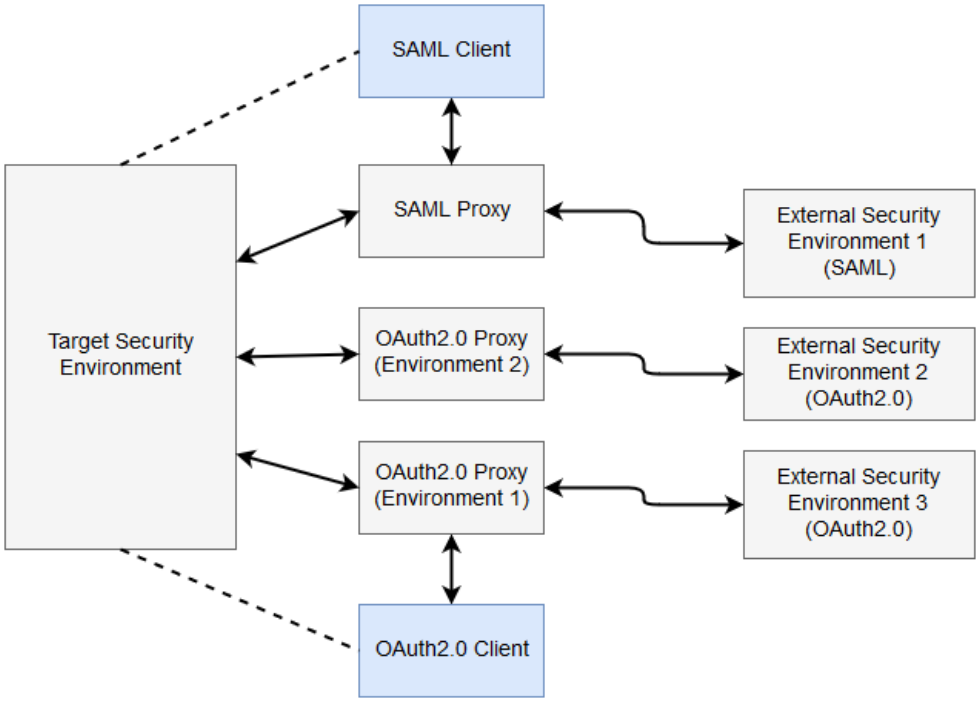

Figure 5. Proxy solutions to enable several Security Environments.

The answer to this question is that the realm of system securitization usually tries to avoid the usage of proxies because of their exploitability [6]. Additionally, it requires a component that implements the logic of where a request should go depending on the type of request, and extends the complexity in terms of the number of components to manage (it is not clear if the proxies belong to the external Security Environments or if they have to be managed by the target Security Environment).

If we assume that this component is the Mediation Service, then this would affect mutual authentication because the user would not be able to authenticate against the authorization server using the X.509 certificate. The only option would be to validate certificates on the Mediation Service directly, which it would represent a clear breach of trust for the certificate.

The preliminary conclusions that arise from this case are that if the mediation service is going to enable mutual authentication, it needs to act as an ad-hoc service to the Authorization Server that acts as the authority of the Security Environment. This means that the service must belong to the Security Environment and that its host should match the host of the Authorization Server (in this case an OpenID Provider).

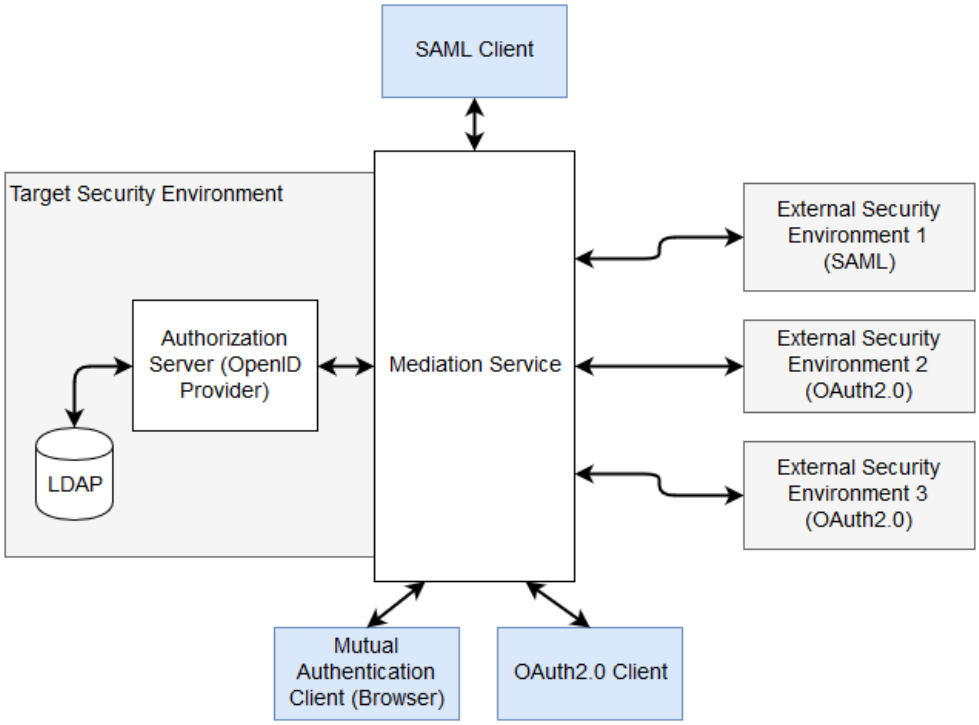

Figure 6. Ad-hoc solutions to enable several Security Environments.

Using this solution, it is only necessary to configure the different internal services of the mediation service in order to communicate with the external identity providers and also to redirect the usual channels of authentication on the Security Environment to those of the mediation service.

7.1.2. Universal tokens

A completely opposite solution to proxy solutions is the idea of "universal tokens" that any Security Environment with the appropriate set of tools would be able to understand and utilize for authentication and authorization purposes. The functionality requirements that come with this solution are:

- Token translators. There needs to be a set of tools for each standard token that allows one to translate them to a universal token without losing information or compromising the integrity of the whole token, that is, leaving a trace of the original token.
- Transitive properties. A token issued by a security environment A needs to be able to become a universal token that then can also be converted to a token understandable by security environment B without losing any of its properties. This leads to the next point.
- Default standard structure. In order for any security environment to be able to get a familiar token out of a universal token, it is necessary that its structure is well known in advance (this links directly to the creation of token translators).
- Keeping trust intact. Even if the translation is possible thanks to a technical solution that solves the previously mentioned requirements (including the provision of signing and encryption functionality), it is still necessary to gather consent of the two parties in charge of the Security Environment management. This would require definition of a common set of user information to be inserted on the universal tokens and also a mapping of each user attribute to the original security standard. Further discussion can be found on the Federated Clouds ER [7].

Although this solution comes as the most complete and ideal, it is nowhere near to being a reality, which means that the Mediation Service will have to exercise a different approach (at least for now).

7.1.3. Passport solutions

In previous sections it has been mentioned that, from a security point of view, proxy solutions should be discarded in order to avoid unnecessary security risks. On the other hand, it is possible to define a component that actually authenticates on behalf of the user and acts as its own entity when interacting with the Authorization Server.

In order to deploy this system, it is necessary to authenticate this component as a client against the server, with their communications and client identity securitized using the client credentials grant (5.2). After this, the component will be able to extract user information from external identity providers and feed that information to the persistence service of the Security Domain, allowing several sources of identity to be matched against a single end-user.

For this kind of solution, passport [8] offers solutions for OAuth2.0, OpenID Connect, SAML and even mutual authentication (X.509 certificates) external providers. This ad-hoc service would be able to configure several clients to the external providers and then extract the user information parameters (only those which are allowed at the origin), populating the internal persistence system (LDAP in the case of this Testbed) for future use.

The following figure shows how the passport service will establish a trusted communication with the Authorization Server to provide user information to the Security Environment. After that, it will manage communications with external identity providers.

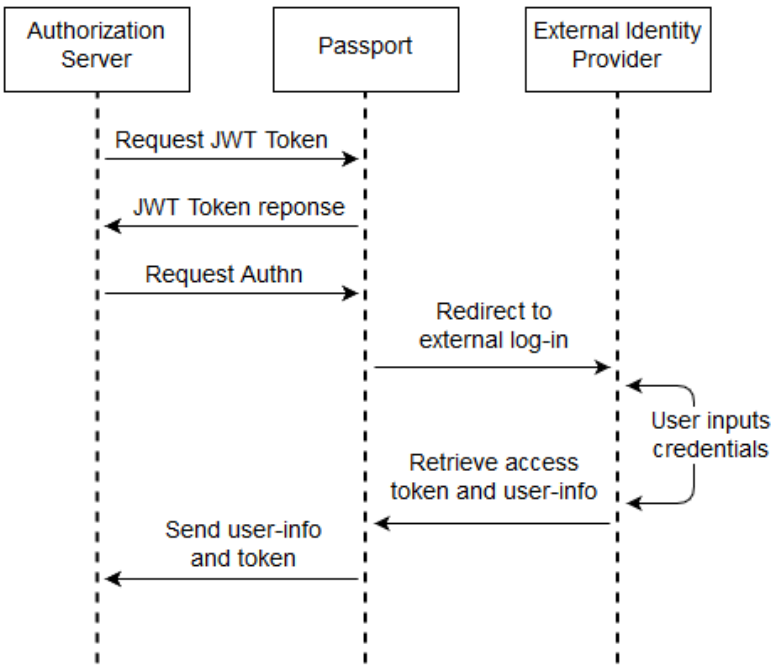

Figure 7. Passport interactors to enable several Security Environments.

In the case of mutual authentication (using X.509 certificates), the passport service will check the browser for potential user certificates that can be used to extract user information. In this case, the access token will not be retrieved.

This solution enables a centralized method for federation that could be used to generate a naïve distributed federation architecture (as depicted on section 9.3).

7.2. Interface context

During this testbed, the architecture proposed on section 6 needed to be modified to adjust to the requirements of the Mediation Service based on passport solutions. The final architecture includes the component and its interactions.

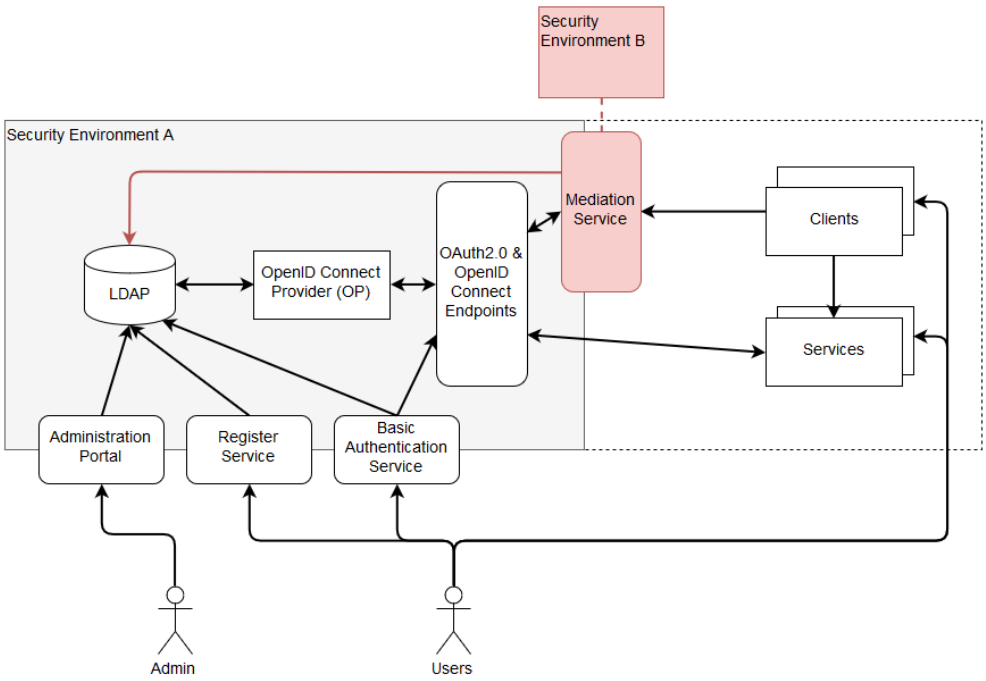

Figure 8. Structure of the modified OpenID Connect security environment.

The inclusion of this component also has an effect over the sequence diagram in Figure 4, modifying the sequence of the interactions with the client resulting in Figure 9 which gives more detail to Figure 8.

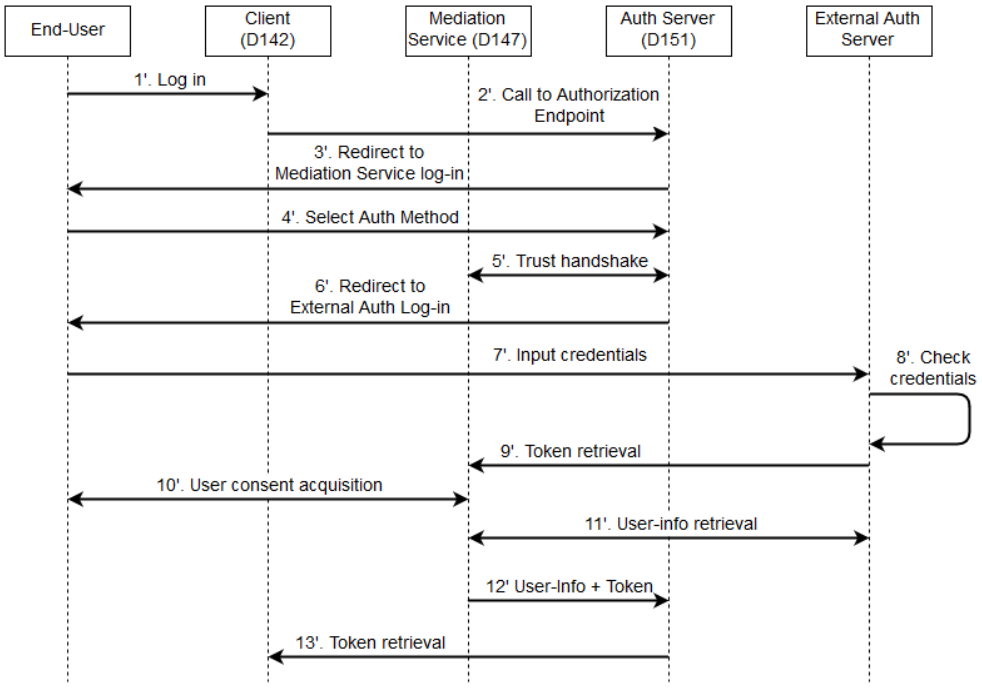

Figure 9. Extended sequence diagram for the interactions with a Mediation Service.

The impact that these changes have on the clients that interact with the Authorization Server and the functionality that becomes available is discussed in the following subsections.

7.2.1. Client-side approach

Taking into account how the architecture has been modified, it is important to allow the client to access the mediation service even when the location or access points are completely unknown beforehand.

To avoid unnecessary extensions of the information that is made available through Capabilities or OpenAPI documents of OGC Web Services, the best design decision is to intercept requests to the Authorization Server and redirect them through the Mediation Service.

When these requests are made using basic credentials, the Mediation Service will simply interact with the available OpenID Connect endpoints. This simplifies implementation on the client side, mainly allowing backwards compatibility.

There is only one critical issue raised by this particular solution, and that is making sure that the communication between the Mediation Service and the Authorization Server is trusted by both entities and is also secured from external risks of impersonation. This is handled on the trust handshake that is performed at the beginning of every mediation transaction.

7.2.2. Interoperability options

From the point of view of the end-user, this mediation service allows the usage of external identity providers that belong to other Security Environments to access services on the domain of the Authorization Server. This means that the user will be able to retrieve valid tokens using credentials that are not initially stored on the Security Environment that needs to be accessed.

During Testbed-14, passport solutions allow access to services through other OAuth2.0

environments such as Google, Facebook, Twitter, LinkedIn and even NextGEOSS (with restricted access to resources). This served as a test case for a full federation between two equal entities and is further explained in section 9.3

On the other hand, this service is easily extensible to the point where SAML environments could also be bundled with the rest of OAuth2.0 given that the necessary attribute mapping between SAML and OAuth2.0 users is put in place.

Finally, passport currently allows access using personal certificates and viability of this solution has also been tested during this Testbed. The resulting interactions are shown in the following figure:

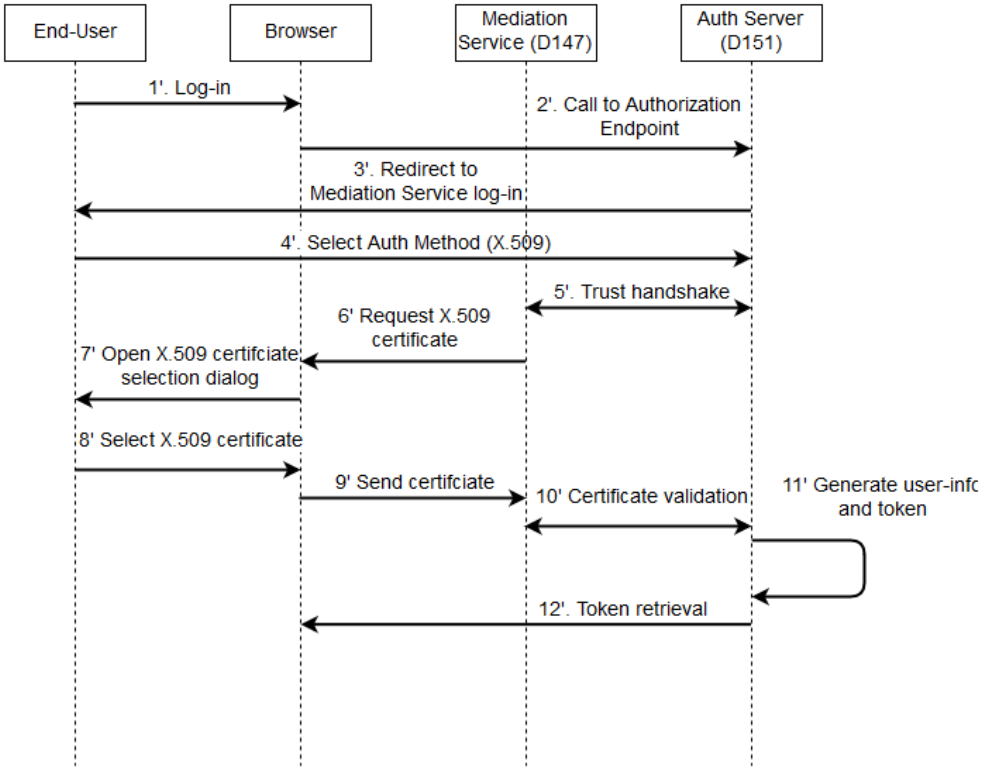

Figure 10. Mediation sequence for X.509 certificates (browser-based).

Even though the available infrastructure does not allow signing of browser-trusted certificates (and this would allow an end-to-end test of these interactions) all steps have been successfully tested separately from the infrastructure (using self-signed certificates). These tests are considered incomplete given that the Authorization Server would have no way of trusting the certificates that the Mediation Service is trying to relay.

7.3. Analysis and Future Work

The Mediation Service has been proven to be unable to act as a WPS since it requires end-user interaction which would not allow complete automatization (automatic HTTP redirections and end-user input would be necessary if state of the art methods are to be used). But this testbed has established that the remaining options are:

- Establish a trust relationship between the user and the mediation service. This is not acceptable by any security infrastructure. It potentially centralizes user credentials outside the security environment.

- Instead of authenticating the end-user, the client is authenticated.

Both options do not really solve the issue, but Testbed 14 has further developed the security understanding of SDI, especially when it comes to workflows. The usage of Dynamic Registration (based on OAuth2.0) and OpenAPI advertising of security requirements will allow for automation of workflows that have secure services requirements.

In order to do so, a CSW Service and a completely new service (separate from the Mediation Service) are necessary. This new service would have to access the CSW catalog containing references to services and their OpenAPI documents. Assuming this OpenAPI document includes references to a security environment and that the referenced authorization server has dynamic client registration capabilities, this new service will be able to dynamically generate a disposable client for each service in a workflow. None of these operations require user interaction or a web browser, and that would allow this component to be a WPS.

The only remaining issue would be traceability of the requests to the end-user, allowing authorization on the service side. That would mean that user authentication and information need to be acquired independently from this new service. The additions to the workflow would be:

- The end-user logs in on each of the required security environments. This list would need to be retrieved based on the workflow and the OpenAPI referenced on the CSW. Once authenticated, a set of JSON Web Tokens would be generated, identifying the user on all environments. These tokens can be retrieved making use of the mediation service (deliverable from this testbed).
- The new service will receive those tokens and use them during the execution of the workflow to validate them and send them to services that perform authorization. Knowing where to validate them or which service requires them would be a matter of consulting the CSW catalog and OpenAPI documents.

This solution still raises some issues regarding governance of the data stored on the CSW service, encryption, and signing of JSON Web Tokens and trust relationships between all components on a workflow.

Chapter 8. Workflow Securitization

NOTE	*BPMN Workflows ER* The BPMN Workflows ER (OGC 18-051) contains further details on Workflows and chaining solutions applied during this Testbed

8.1. Design Overview

There are several points that need to be clarified during this Testbed in order to properly design security solutions for workflows and these are:

- The security environment that encloses the workflow manager and BPMN engine
- Discoverability of resources and their security constraints
- Application of granular access to resources when security requirements diverge
- Secured relaying of user information and output data through the workflow

In the case of this Testbed, all components belong to the same security environment, which simplifies the issue of discoverability, and relaying of outputs and user information (Figure 9).

In the case of discoverability, the usage of extended constraints on the services to identify which Security Environment the service belongs to (combined with the usage of a CSW service that catalogs references to all services) allows one to determine when a service expects a security token.

8.2. Security analysis and Future Work

Using the information extracted from security constraints, it is possible for the BPMN engine to decide if a security token will be sent to the service (by encoding it on the BPMN document only in the specific requests), allowing external services from other security environments to be accessed without concern of sharing user information unintentionally.

In the case of diverging security requirements from specific resources this Testbed raised some concerns on specific use cases that involve chaining unsecured services with secured services, mainly when outputs from a secure service are not meant to be shared with unsecured services (or just services outside the Security Environment). To solve this issue, there are several possibilities to be analyzed:

- The outputs are stored on a Resource Server that has protected access and inputs to the unprotected service are passed as a reference, forcing it to also act as a client that needs to be authenticated.
- The workflow manager (or a mediation component) has access to additional security constraints not only for access but also for output discoverability and is trusted to enforce them, not allowing chains that are against these constraints.

Figure 11 shows an example of the interactions described on the first point. This would be the best option in terms of governance, meaning that the centralized authority would still be the authorization server, but at the same time requires services to authenticate themselves in order to

access outputs from other services and this would result in increasingly complex development work.

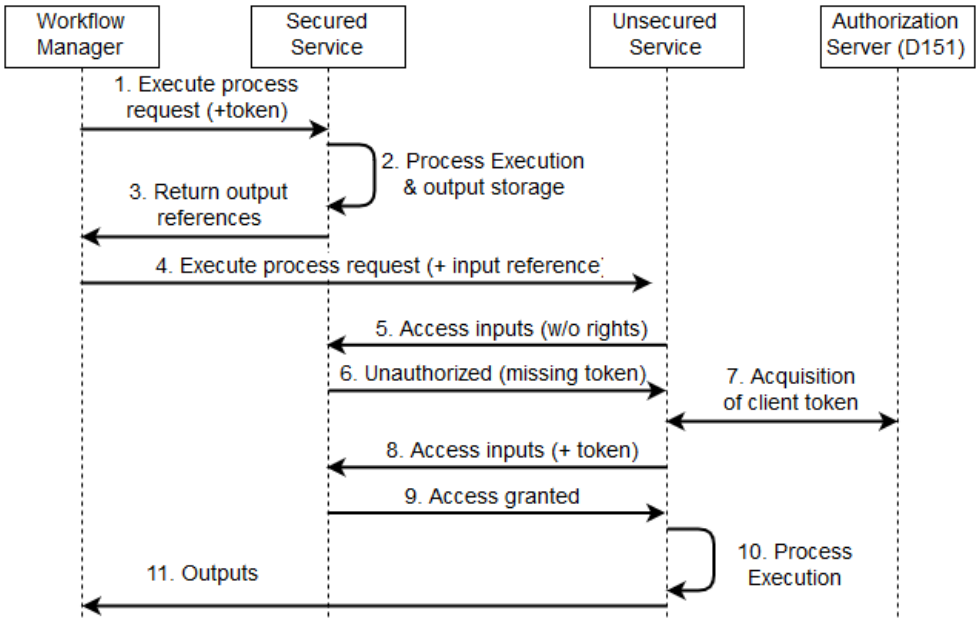

Figure 11. Sequence diagram for Resource Server based protection.

This methodology would require any services to also act as clients that acquire tokens to extract outputs that come as references, but could also be against best-practices for OGC services and workflows.

Trusting the workflow manager (or a mediation service) would be essential in order to apply the second option, meaning that each workflow manager would have to be clearly authenticated and authorized to execute workflows. In order to do this, secured services would expect a way of authenticating the workflow manager itself in addition to the end-user authentication and authorization that is already taking place. This interaction is shown on Figure 12, assuming that the business logic behind the chain is valid when considering the security constraints. It would only require the Workflow Manager to be authenticated (act as a client) instead of having external services be modified in order to be compatible.

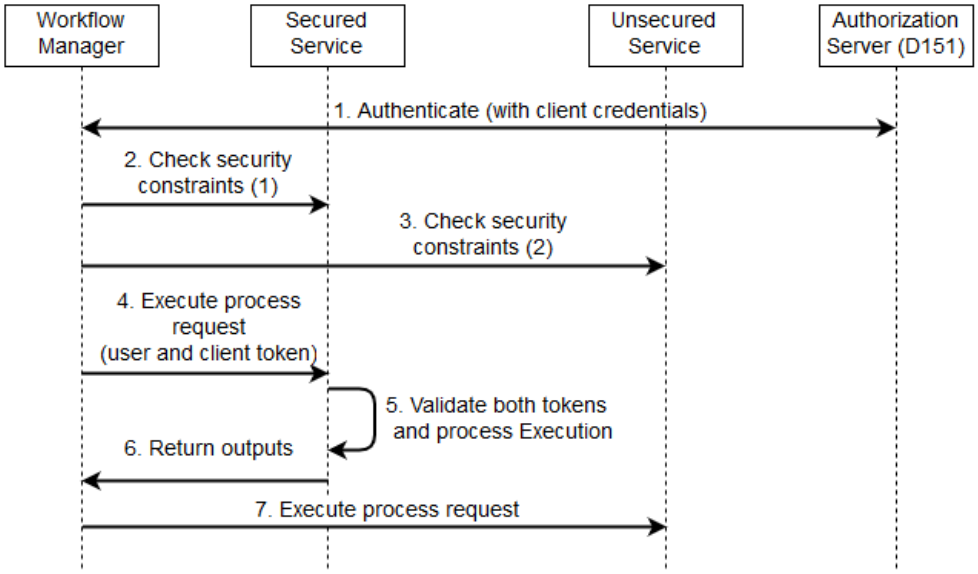

Figure 12. Sequence diagram for Workflow Manager based protection.

Any of these solutions can be further extended with the usage of business logic on the BPMN validation phase, as explained on the Workflows ER.

Chapter 9. Federated Clouds Securitization

NOTE	*Federated Clouds ER (OGC 18-090)* This subsection will include a reference to gap analysis discussion performed on the Federated Clouds ER (OGC 18-090). All sections should be read in the context of the aforementioned ER.

9.1. Preliminary Analysis

In the topic of Security, several domains can be identified that usually cover all needs required by any organization or individual. While this Engineering Report has focused mainly on Access Control mechanisms, there is a wide variety of topics (such as Risk Assessment or Threat intelligence, User Education…) that are usually considered important aspects of security. Although this Testbed does not cover all of them (focusing more on Security Architecture), the concept of Cloud Security and Federated Identity are nevertheless mentioned in several Engineering Reports.

In that aspect, Federated Identity becomes a strong relationship that can be perceived between Federated Clouds and the Security Architecture. During this Testbed, the Federated Clouds ER (OGC 18-090) offers a more formal approach to the topic.

NOTE	*Definitions* When applied to Federated Clouds, the term "Security Environment" becomes deprecated in favor of "Administrative Domain" which implies a different approach to Identity Management in the context of Cloud Federation. Detailed definitions can be found on Section 3.

On the other hand, this ER offers a technical approach that is not meant to follow best practices or to conform with the current state of the art extracted from federated clouds architectures. This ER instead serves as a starting point that can be analyzed, modified and extended in order to adjust its technical capabilities to the needs identified during this Testbed.

Initially, the combination of both D151 Authorization Server and D147 Mediation Service establishes what is known as a Centralized Federation figure that allows other Administrative Domains to populate its persistence systems with information from external users.

At the end of this Testbed, this centralized federation is connected to another centralized federation making them both act together as a distributed federation that requires a series of discovery policies and governance solutions in order to become feasible outside the scope of the Testbed.

9.2. Centralized Design Overview

The architecture proposed for this design is basically the one depicted in Figure 8.

This system allows users from several domains to access Testbed services using their domain credentials instead of credentials stored inside the Authorization Server. The amount of user information collected by the Authorization Server is restricted by the Authorization services that

reside in the external Administrative Domain and needs to be agreed upon when configuring the Mediation Service.

With this solution, the authentication of end-users is delegated to external entities, but authorization for the access to the internal service must be performed utilizing the aforementioned user information.

9.3. Federated Design Overview

When another instance of the current Testbed architecture is equipped with a Mediation Service that also allows delegation of authentication on its side, it is possible to define a custom client on each side that serves as a relay between both environments.

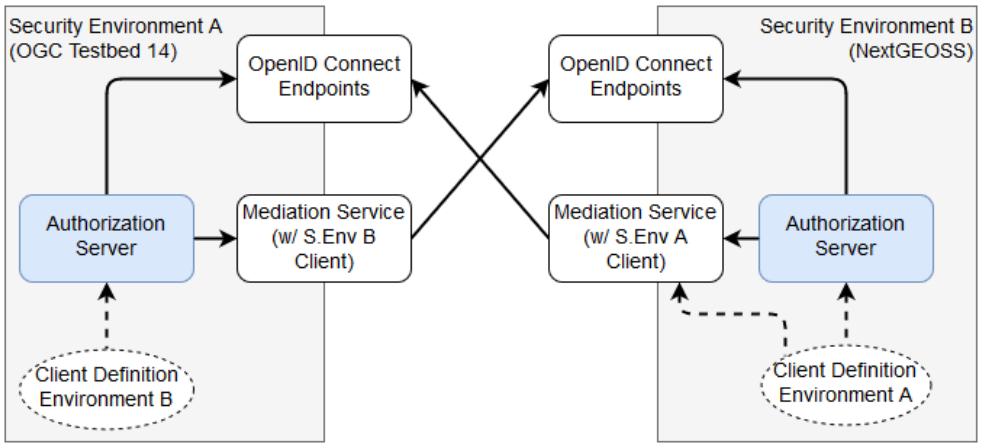

Figure 13. Basic diagram for a two-way federation.

Each of the environments has the capabilities of defining which scopes (and corresponding attributes) can be requested by the client defined in the opposite Security Environment (enclosed on an Administrative Domain) and it can also define its own client to persist only relevant data or to map an already existing user to an authentication request coming from outside of the Administrative Domain.

9.3.1. NextGEOSS Federation

NextGEOSS is the next generation of the Global Earth Observation System of Systems with the main goal of providing a federated data hub for Earth Observation data. Fully sponsored by the European Commission (EC), the main goal of its User Management platform was to create a GEOSS User Management system that provides users with Single-Sign-On (SSO) access to GEOSS data and services in a federated environment.

Its underlying technical aspects are very similar to those displayed by the security deliverables developed during this Testbed and it is a clear candidate for the proposed federation architecture. During this Testbed, an architecture similar to the one shown on Figure 13 has been put in place. In the case of OGC Testbed-14, no restrictions to user information acquisition are really necessary since there is no operational use envisioned for the system.

In the case of NextGEOSS, the platform defines a series of attributes under a specific scope that is not reachable. This scope contains operational parameters that will never be enclosed in the tokens

released when interacting with the Authorization Server defined in this Testbed, even if Testbed clients request this scope. On the other hand, basic user profile information can be easily shared between both Administrative Domains as a result of the work performed during this Testbed.

9.4. Analysis and Future Work

The federated solution proposed in this Testbed utilizes current state-of-the-art technologies such as OAuth2.0 / OpenID Connect combined with Passport solutions in order to provide a simple federation mechanism with specific constraints.

Although this architecture serves as a way of providing governance over user information, it lacks the ability to define discovery governance and policies for the services that reside on the Administrative Domain itself.

The main solution to this requirement is to extend OAuth2.0 capabilities so services can act as end-user entities that can register and protect its own resources. For this specific purpose, the UMA standard (User Managed Access) allows the definition of an additional OAuth2.0 flow that specifies how these kinds of transactions can be performed and how different access policies can be enforced on granular sets of resources.

Additionally, it is important to understand the difference between a Security Environment (exercised during this Testbed) and an Administrative domain. The latter is widely explained in the Federated Clouds ER (OGC 18-090) and serves as a wide understanding of what a Security Environment requires in order to comply with the most common governance and discoverability issues when performing federated identity management.

Chapter 10. Technology Integration Experiments

NOTE	*Summary* This section will include a summary of all the Testbed components involved in the security architecture described in previous sections and their security-relevant connectivity and functional tests.

10.1. Components

Given the interactions mentioned in Section 6.2, the following Testbed TIE components perform security interactions:

Name	Deliverable ID	Endpoint Samples
Authorization Server	D151	https://testbed-14-sso.elecnor-deimos.com
Client Implementation (GISFCU)	D142	https://map.gis.tw/testbed14/
Client Implementation (Solenix)	D143	-
Service Implementation - WFS3.0 instance (Interactive Instruments)	D140	https://services.interactive-instruments.de/t14/wfs3/daraa-sec/ & https://services.interactive-instruments.de/t14/wfs3/zaatari-sec/
Service Implementation - WFS3.0 instances (GeoSolutions and CubeWerx)	D113	E.g. https://tb14.cubewerx.com/cubewerx/cubeserv/default/wfs/3.0.0/Daraa & https://cloudsdi.geo-solutions.it/geoserver/daraas/wfs3
NGA Analysis Tool WPS (52 North)	D146	http://testbed.dev.52north.org/data-quality-wps/WebProcessingService?Request=GetCapabilities&Service=WPS
Mediation Service	D147	[Integrated on D151] https://testbed-14-sso.elecnor-deimos.com

10.2. Pairings

The following Security TIE tests have been performed:

Components	Use Case	Result
D151 / D142	Authentication based on Authorization Code / Resource Owner / Implicit Flows	OK
D151 / D143	Authentication based on Authorization Code Flow	OK
D151 / D140	Token validation and User-Info release	OK
D151 / D113	Token validation and User-Info release	OK
D151 / D146	Token validation and User-Info release	OK
D147 / D151	Token acquisition from external Authorization Servers (NextGEOSS/Google/Facebook/LinkedIn/Twitter)	OK

10.3. Detailed tests

For further detail on TIE experiments, including multimedia samples and a matrix per component, the following sources can be consulted:

- D021 - Secure Resource Oriented Geospatial Data Handling ER contains detail on specific Authentication Grants on client and service side.
- D026 - Workflow ER contains details on token relaying experiments.

Appendix A: Integration Guide

A.1. General view of authentication procedures

A.1.1. Authentication Flows

In order to interact with this service, it is necessary to implement one of the three authorization flows defined by OpenID Connect 1.0 standard: implicit flow, authorization code flow and hybrid flow.

- Implicit flow. This method is recommended for browser-based apps (such as Deliverable D143). Its main steps are:
  1. A request is made to the Authorization Endpoint. The Authorization Server will redirect the user to the sign-in page.
  2. The end-user will then authenticate with a set of required credentials.
  3. The Authorization Server will answer back with a redirection URI and an Access Token.
  4. The Access Token can be used to request information about the end-user via the User Info Endpoint.
- Authorization Code flow. This method is recommended for apps running on a web server or native mobile applications. Its main steps are:
  1. A request is made to the Authorization Endpoint. The Authorization Server service will redirect the user to the sign-in page.
  2. The end-user will then authenticate with a set of credentials required.
  3. The Authorization Server will answer back with an authorization code.
  4. The Client can now use the received code to request an Access Token through the Token Endpoint.
  5. Once the client application has acquired an Access Token, it will be possible to request information about the end-user via the User Info Endpoint.
- Hybrid Code flow. This method merges characteristics from both of the previously mentioned methods. Currently, the usage of this method is not recommended with the Authorization Server.

To implement any of the flows, it is necessary to specify the response type on the request to the Authorization Server. OpenID Connect specification indicates the combination of response types necessary to implement each flow:

"response_type" value	Flow
code	Authorization Code Flow
id_token	Implicit Flow
id_token token	Implicit Flow
code id_token	Hybrid Flow
code token	Hybrid Flow
code id_token token	Hybrid Flow

Figure 14. Response Types on OpenID Connect

A.1.2. Client authentication

When accessing the Token Endpoint, clients using the implicit flow are not required to be authenticated, but if the client uses the Authorization Code Flow, it must provide its credentials.

These credentials will be provided by Deimos: client_id and client_secret.

A.1.3. Request Endpoints for Authentication

All endpoints require a set of mandatory parameters in order to generate a valid response. Their URLs can be obtained by means of a Discovery URI that answers back with a set of endpoints and their URLs. All these URLs must be accessed by means of a GET request, and require a set of mandatory parameters:

- **Discovery URI**: /.well-known/openid-configuration
- **Authorization Endpoint** (GET): /oxauth/restv1/authorize Parameters:
  - scope: The request should include an array of scopes, with one of them being “openid”.
  - response_type: Its value should be desired combination according to the OpenID Connect response type table.
  - client_id: Provided by Deimos.
  - redirect_uri: It should point to the client application and match the URI given on the acceptance request.
  - state (optional): Opaque value used to maintain state between the request and the callback. Typically, Cross-Site Request Forgery (CSRF, XSRF) mitigation is done by cryptographically binding the value of this parameter with a browser cookie.
  - nonce (optional/**required for Implicit Flow**): String value used to associate a Client session with an ID Token, and to mitigate replay attacks (it can have ANY value)

Example:

```
GET
oxauth/restv1/authorize?scope=openid&client_id=@!5C7F.E36B.5DE3.15EE!0001!6B53.87B4!00
08!A121.D32B.8BCD.4E14&redirect_uri=app://test&response_type=code
```

If the application is trying to authenticate without user input, user credentials must be provided through the Authorization header. The code will be encoded as a parameter in the Location response header.

- **Token Endpoint** (POST): /oxauth/restv1/token
  - grant_type: implicit / authorization_code (depending on the authorization flow).
  - Code*: Used only with grant_type=authorization_code
  - redirect_uri: It should point to the client application and match the URI given on the acceptance request.
  - scope: The request should include an array of scopes, with one of them being “openid”.
  - client_id*: Provided by Deimos, only necessary with grant_type=authorization_code.
  - client_secret*: Provided by Deimos, only necessary with grant_type=authorization_code

A.2. Implementation on the Client side

A.2.1. Static Registration

In order to allow a client application to delegate its sign-in function under the SSO system of the Authorization Server, it is necessary to provide the following parameters:

- Application Type: An application could be either NATIVE or WEB. [1: In this case it would be WEB]
- Policy and ToS URI: These resources contain the application policies regarding the usage of user personal information. [2: Not necessary for the purposes of this demonstration]
- Redirect Login/Logout URI: Only the first is mandatory. Indicates the URL or App Link where the sign-in service will redirect users after login. [3: The logic implemented on this webpage should retrieve the token from the URL]
- Required OAuth2 Scopes: These scopes indicate which kind of information and access the Client Application is able to grant to users. [4: OpenID scope is mandatory (but its use is optional) and ogc_user is default for this system]

After the application has been approved and configured, the following parameters, necessary to connect to the SSO service, will be provided to the client:

- Client ID: Unique identification sequence for your client.
- Client Secret: Necessary to perform Authentication on the Token Endpoint.

For the purposes of this OGC Testbed-14 task, client authentication (in addition to end-user authentication) is not mandatory but could serve as a proof-of-concept for mutual authentication. Client credentials can be passed either as an Authorization header (encoded as Basic) or in the form of the POST request. Only one of these options can be enabled at the same time for each client.

A.2.2. Dynamic Registration

NOTE	*Dynamic client expiration date* During Testbed-14 all dynamically generated clients will have 1 hour of validity and after that they will be completely removed.

Another way of registering clients is through the dynamic register method defined on the OAuth2.0 specification and extended to allow OpenID Connect parameters. This method allows the client to automatically register itself, receiving all the necessary parameters in order to integrate with the security service. Clients registered using this method will inevitably have an expiration date and its usage implies the necessity of restricting the use of the functionality to avoid exploits or security breaches.

In order to register a new client, a call must be performed to the register API Endpoint. The only mandatory parameters are **redirect_uris** and **client_name**. The rest of the parameters on this request are completely optional, which means that the Authorization Server will fill in values by default. In any case, the client can initially create a client with default configuration and update it afterward using methods described in this section.

- **Register Endpoint** (POST): /oxauth/restv1/register
  - redirect_uris (mandatory): array of strings containing the redirect_uris for the client
  - client_name (mandatory): string containing the client name
  - response_types: array of strings containing the desired response types
  - grant_types: array of strings containing the desired grant types
  - application_type: can either be "web" or "native"
  - subject_type: can either be pairwise (each user is assigned a unique "sub" parameter) or public
  - token_auth_method: identifies the authentication method when retrieving tokens from the Token Endpoint
  - default_max_age: indicates the max age for tokens obtained using this client
  - default_acr_values: there is no need to assign any value to this parameter, but if set to "passport", it will redirect the user to the mediation service.
  - [Other parameters]

There is a wide variety of parameters that can be configured on this request and all of them can be checked on the discovery document:

Disovery document JSON:

```
GET /.well-known/openid-configuration
```

For example, if the client wants to use signed JSON Web Tokens, it is necessary to include the corresponding parameter in the call to the Register Endpoint. The discovery document has a field named "id_token_signed_response_alg_values_supported" with several signing methods. One of them can be selected by the client and passed through the parameter "id_token_signed_response_alg".

There is only ONE parameter that is not configurable through this dynamic method and that is the "scopes" parameter. These are filled in by the authorization server for security purposes based on the parameters received on the request. Testbed-14 behavior will be to assign the same limited scopes to all dynamic clients, but it is possible to restrict scopes based on domain, grant_types or

any other parameter.

Any call to the Register Endpoint will have the Authorization Server answer back with a JSON document that indicates metadata about the recently generated client. In particular, there will be two additional fields that can be of use to the client and those are:

- **registration_access_token**: a bearer token that allows the requester to list or modify metadata about a specific client
- **registration_client_uri**: a URI assigned to a specific client for further interactions

There are mainly two more actions that can be performed with these parameters:

Client metadata retrieval:

```
GET <registration_client_uri> -H "Authorization: Bearer registration_access_token"
```

Client metadata update:

```
PUT <registration_client_uri> -H "Authorization: Bearer registration_access_token" +
JSON_BODY
```

Basic example

The most basic example for dynamic client registration would be:

Example:

```
POST /oxauth/restv1/register
{
         "redirect_uris": [
              "https://client.example.org/callback",
              "https://client.example.org/callback2"
          ],
         "client_name": "Basic Client"
}
```

The Authorization Server would respond with the following information (example):

Example of a registration response:


```
{
  "client_id": "@!27B7.E085.07A1.6DE7!0002!F5E4.0B8E!0008!C14A.232C.E89C.C514",
  "client_secret": "b2a5fc13-3593-4100-8287-db844b4845f2",
  "registration_access_token": "dee762cf-b134-4e2b-81fd-1238c9299135",
  "registration_client_uri": "https://testbed14-sso.elecnor-
deimos.com/oxauth/restv1/register?client_id=@!27B7.E085.07A1.6DE7!0002!F5E4.0B8E!0008!
C14A.232C.E89C.C514",
  "client_id_issued_at": 1533812916,
  "client_secret_expires_at": 1533816516,
  "redirect_uris": [
    "https://client.example.org/callback",
    "https://client.example.org/callback2"
  ],
  "response_types": ["code"],
  "grant_types": [
    "authorization_code",
    "refresh_token"
  ],
  "application_type": "web",
  "client_name": "Basic Client",
  "subject_type": "pairwise",
  "id_token_signed_response_alg": "RS256",
  "token_endpoint_auth_method": "client_secret_basic",
  "require_auth_time": false,
  "frontchannel_logout_session_required": false,
  "scopes": [
    "openid",
    "uma_protection",
    "permission",
    "user_name",
    "email",
    "profile"
  ]
}
```


Behavior by default is to utilize the Authorization Code grant, allowing refresh tokens, and the default set of scopes can be seen in the example. The client secret has also been randomly generated.

End-User clients

```
POST /oxauth/restv1/register
{
      "redirect_uris": [
        "https://client.example.org/callback",
        "https://client.example.org/callback2"],
      "client_name": "D142 Client",
      "token_endpoint_auth_method": "client_secret_post",
      "response_types": ["token", "id_token", "code"],
      "default_acr_values": ["passport"],
}
```

The "default_acr_values" set to "passport" allows the service to be redirected through a mediation service (with federation options).

Testing client (for service implementers)

```
POST /oxauth/restv1/register
{
      "redirect_uris": [
        "https://client.example.org/callback"],
      "client_name": "WFS Service Example",
      "token_endpoint_auth_method": "client_secret_post",
      "grant_types": ["password"]
}
```

A.2.3. Implementation Solutions for JS Clients

For Testbed-14 web-based clients, there are several Free and Open Source JavaScript solutions available that could implement the implicit flow. In general, all of them perform a call against the Authorization Endpoint:

- **Authorization Endpoint** (GET): /oxauth/restv1/authorize
  - scope: “openid ogc_user”.
  - response_type: “id_token token”.
  - client_id: Provided by Deimos.
  - redirect_uri: <TBD>

Example:

```
GET
/oxauth/restv1/authorize?scope=openid%20ogc_user&client_id=<TBD>&redirect_uri=<TBD>&re
sponse_type=id_token%20token
```

A.2.4. Implementation Solutions for Clients with Back-end

For Testbed-14 back-end powered clients, there are several Free and Open Source solutions available that could implement the authorization code flow. In general, all of them perform a call against the Authorization Endpoint to retrieve a code and then exchange it for a token on the Token Endpoint:

- **Authorization Endpoint** (GET): /oxauth/restv1/authorize
  - scope: “openid ogc_user”.
  - response_type: “code”.
  - client_id: Provided by Deimos.
  - redirect_uri: <TBD>

Example:

```
GET
/oxauth/restv1/authorize?scope=openid%20ogc_user&client_id=<TBD>&redirect_uri=<TBD>&re
sponse_type=code
```

- **Token Endpoint** (POST): /oxauth/restv1/token
  - scope: “openid ogc_user”.
  - grant_type: authorization_code.
  - code: Obtained on the previous request.
  - client_id: Provided by Deimos.
  - client_secret: Provided by Deimos
  - redirect_uri: <TBD>

Example:

```
POST /oxauth/restv1/token -d
'scope=openid%20ogc_user&client_id=<TBD>&client_secret=<TBD>&redirect_uri=<TBD>&grant_
type=authorization_code&code=<CODE>
```

A.3. Implementation on the Service Side

A.3.1. Token Validation and End-user Authorization

Use this Endpoint to acquire user information:

- **User-Info Endpoint (GET)**: /oxauth/restv1/userinfo
  - access_token: Acquired via Token or Authorization endpoints.

Example:

```
GET oxauth/restv1/userinfo?access_token=<TOKEN>
```

NOTE

The "sub" parameter

When using the openid scope, the Authorization Server will always answer with a "sub" parameter that is supposed to identify an End-User with a unique string. This will only be the case for the Authorization Grants mentioned in this Integration Guide. Other flows might collapse all subs into the client_id to avoid unintentionally leaking user information with non-OpenID grants.

A.3.2. Service-Side Authorization

Any Service should define which user attributes are required to interact with its services. After that, any received request must contain an access token (obtained using the Authorization/Token Endpoint).

This access token can be included in an internal request to the Authorization Server User Info Endpoint, which returns the set of default attributes that can be checked internally by the service in order to determine if access should be granted.

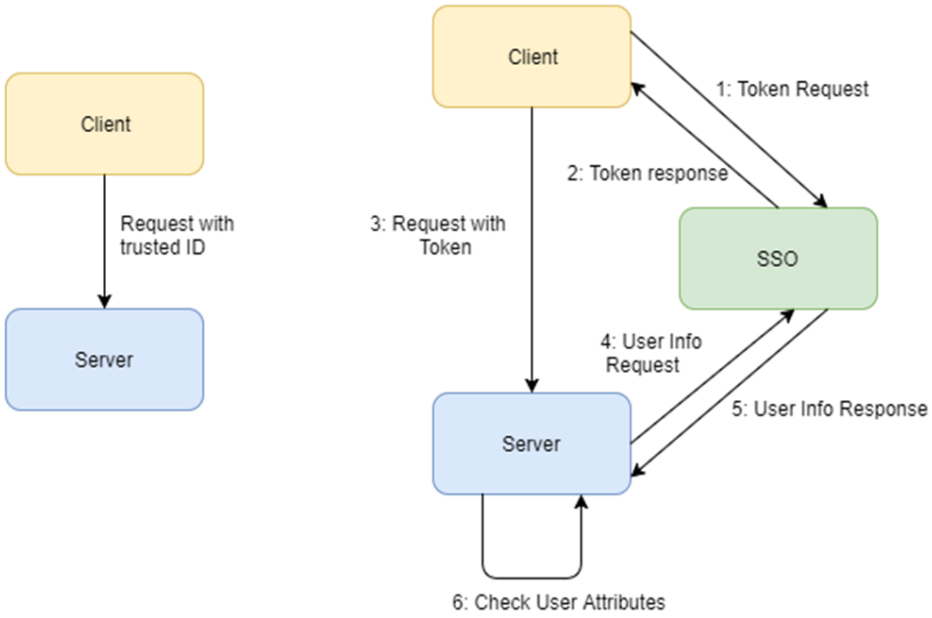

Figure 15. Service-Side Authorization example

In this image, the Server does not know if the Client is the one performing the Token Request. For Testbed-14 the Token Request is performed on the Client (D143) and relayed to the BPMN Engine that will act as the client when providing the token.

A.4. Resources for developers

A.4.1. Libraries and documentation

In order to develop a Client Application that satisfies the needs specified by OpenID Connect 1.0 and

Testbed 14 SSO, specifications can be found here:

- OAuth2.0: https://oauth.net/2/
- OpenID Connect: http://openid.net/specs/openid-connect-core-1_0.html

All the information provided in the previous sections combined with the specifications of the standards involved in this service can be more easily implemented through available libraries dedicated to each platform and environment:

- Single Page Applications (SPA, usually based on JavaScript) [https://github.com/IdentityModel/oidc-client-js]
- Native Apps for iOS and macOS [https://github.com/openid/AppAuth-iOS]
- Native Apps for Android [https://github.com/openid/AppAuth-Android]
- Reverse Proxy Solutions [https://github.com/pingidentity/mod_auth_openidc]
- Oauth2.0 plugin for JIRA [https://marketplace.atlassian.com/plugins/com.miniorange.oauth.jira-oauth/server/overview]
- Web applications:
  - Django rest framework for OAuth2.0 [https://github.com/PhilipGarnero/django-rest-framework-social-oauth2]
  - Python social Auth [http://python-social-auth.readthedocs.io/en/latest/]
  - Django OAuth Toolkit [https://django-oauth-toolkit.readthedocs.io/en/latest/]
  - AngularJS [https://github.com/manfredsteyer/angular-oauth2-oidc]

These libraries contain documentation and samples that implement a basic Client App that can be used for authentication.

Appendix B: Revision History

Table 1. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
May 22, 2018	H. Rodriguez & J. Doval	.1	all	Initial ER
July 27, 2018	H. Rodriguez & J. Doval	.2	all	Sample content
September 28, 2018	H. Rodriguez & J. Doval	.3	all	Preliminary DER
October 30, 2018	H. Rodriguez & J. Doval	.4	all	Near-Final DER
November 21, 2018	H. Rodriguez & J. Doval	.5	all	Final DER

Appendix C: Bibliography

1. Kantara Initiative: UMADef User Managed Access Specification, https://docs.kantarainitiative.org/uma/ed/uma-core-2.0-01.html, (2016).
2. IETF OAuth Working Group: OAuth 2.0 Implicit Grant, https://oauth.net/2/grant-types/implicit/.
3. IETF OAuth Working Group: OAuth 2.0 User Authentication, https://oauth.net/articles/authentication/.
4. A JSON Token-Based Authentication and Access Management Schema for Cloud SaaS Applications. In: Future Internet of Things and Cloud (FiCloud). IEEE. pp. 47–53 (2017).
5. Matheus, A.: OGC Web Services Security Standard. OGC 17-007r1,Open Geospatial Consortium, http://docs.opengeospatial.org/is/17-007r1/17-007r1.html (2019).
6. Durumeric, Z., Ma, Z., Springall, D., Barnes, R., Sullivan, N., Bursztein, E., Bailey, M., Halderman, J.A., Paxson, V.: The security impact of HTTPS interception. Presented at the (2017).
7. Lee, C.A.: OGC Testbed-14: Federated Clouds Engineering Report. OGC 18-090r1,Open Geospatial Consortium, https://docs.opengeospatial.org/per/18-090r1.html (2019).
8. Passport, http://www.passportjs.org/.