Securing Integration of Cloud Services in Cross-Domain Distributed Environments

Bojan Suzic
Institute for Applied Information Processing and Communications
Graz, Austria
bojan.suzic@iaik.tugraz.at

ABSTRACT

Traditional cloud integration scenarios, as adopted by many organizations, assume business processes to be executed in a cross-domain context, connecting on-premise and cloud applications. The emerging model of cloud-based integration platforms extends these scenarios by transferring business process execution entirely to the cloud. Although this approach provides numerous benefits and opens a new range of opportunities, its adoption requires reconsideration of currently applied practices and their adjustment to a new perspective.

In this work, we analyze the existing approaches to cross-domain service composition based on cloud integration platforms. We particularly focus on the security of these approaches, considering currently dominant OAuth 2.0 web authorization protocol and emerging UMA protocol. For this purpose, we present a new tool that enables UMA support in Apache Camel integration framework. We then analyze and discuss the integration flows relying on both protocols. Finally, based on RMIAS framework, we provide a security assessment of both approaches, presenting an overview of issues and challenges for future work.

CCS Concepts

•**Networks** → *Cloud computing;* •**Security and privacy** → *Distributed systems security; Web protocol security;*

Keywords

web protocols; integration platforms; data security; cloud computing; service composition

1. INTRODUCTION

SAC 2016,April 04 - 08, 2016, Pisa, Italy

ACM 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851622

The cloud adoption among the enterprises has already taken a significant scale. Considering recent surveys [4], 86 percent of companies globally spend at least part of their IT budgets on cloud services. However, as diverse as they are, cloud services can be adopted in various ways, enabling hybrid scenarios that allow a vast amount of critical business services to remain executed in an on-premise setting. Such approaches enable enterprises to improve their business processes by leveraging cloud services, but still provide enough options to restrict organizational exposure to security and integration issues that appear with new platforms.

The emerging category of cloud-based integration services, also known as *Integration Platform as a Service* (IPaaS), as promising as it is, aspires to disrupt the standard model of hybrid on-premise and cloud-integrations by transferring the execution of core business processes solely to the cloud.

In this work, we approach the issue of cloud-based integrations from the perspective that considers data security and privacy aspects in cross-domain based interactions and flows. We analyze the applicability of existing and emerging protocols and their impact on security in complex environments. In our analysis, we point to the issues derived from the existing setups and interfaces and demonstrate how a novel, user-centered and decentralized authorization protocol can be applied to improve the security of cloud integration processes. However, we show that even this solution might not satisfy all security requirements in an optimal way.

1.1 Contribution

Our contribution is threefold. First, we present a new tool, *acUMA*, an extension of Apache Camel integration framework that enables the reliance on novel UMA [16] protocol for cloud-based authorization and service integration. This is, by our knowledge, the first implementation of such service. In our second contribution, we apply this tool and analyze the flows of both OAuth 2.0 and UMA protocols in the context of cloud integration platforms. Based on this analysis, we provide a discussion and comparison of these two approaches in the terms of the controls that enhance security. Our last contribution in this work is security assessment of both approaches based on RMIAS framework [3], which establishes an overview of security-related challenges and issues, mapping the directions for future work to advance security aspects of cloud-based integrations.

2. BACKGROUND AND RELATED WORK

The approach of cloud-based integration got broader attention recently, as the products focused on integration and management of cloud services started to appear and gain traction. The emergence of these services, however, does not imply the establishment of a new discipline. Enterprise integration, in its various forms, has been present for more than a decade [24]. Following its emergence in the form of cloud-based technologies, analysts tried to establish and define the field of cloud-based integration services. One of the notable contributions in this direction has been provided by Pezzini et al., who identified IPaaS as a suite of cloud services enabling *development, execution and governance of integration flows connecting any combination of on-premises and cloud-based processes, services, applications and data within individual, or across multiple organizations* [17].

The analysis of functional and organizational aspects, as well as the detailed overview of integration platforms, technologies and challenges have been provided in [19]. Pethuru identified challenges for SaaS and XaaS integrations, including dynamic nature of SaaS interfaces, dynamic characteristics of metadata of SaaS solutions and *data quality and integrity issues*. Potocnik and Juric provided other classification of issues in SaaS as IPaaS integration, including *data integrity and security*, *data transformation and migration*, *connectivity*, *governance*, *monitoring* and *orchestration* [18]. Other contributions in defining the concepts and challenges in cloud-based service integrations have been provided in the works of Kleeberg et al. [12] and Baude et al. [2].

As they both deal with the integration of enterprise systems and processes, integration platforms often overlap or share similar issues with other related concepts. These include *Cloud Brokerage*, *Business-to-Business Integration*, as well as *Enterprise Application Platforms* and *API Management*.

2.1 Integration Platform Scenarios and Issues

The typical activity performed under integration platforms in the cloud is presented in Fig. 1. It shows the execution of workflows in the cloud integration platform that consolidate the services and resources present across the cloud. In this figure, the integration platform additionally connects to on-premise organizational systems, but its processes can also stretch to the systems of other organizations, showing the capability to access heterogeneous, cross-domain systems.

One of the scenarios for the interest of this work considers the access to the customer's resources located at other cloud services, as shown in the figure. The example instance of this flow can be illustrated with the platform that connects to organizational *Gmail* account, retrieves the messages, processes them internally and then, according to predefined triggers, consumes the interface on organizational *Salesforce* account. This scenario illustrates the cloud-based execution of a business process that consumes customer's resources across different cloud instances, which will be further dealt with the scope of this work.

The typical integration scenario does not differ much from the previous example. In its base form, it encompasses the use of organizational accounts at third party providers, with the goal to execute the predefined tasks or complex workflows. This processing is commonly realized using *Web APIs* exposed by the service provider, which are secured using widely adopted mechanisms. Based on the current state of the web, one of those mechanisms is OAuth 2.0 protocol [9]. OAuth 2.0, however, lacks fine-grained, policy-enhanced, assured and auditable data flow control and monitoring, as it will be shown in this work.

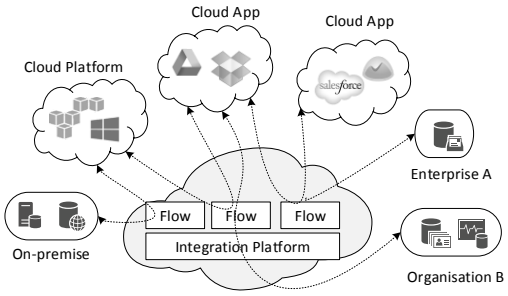

Figure 1: Flows in cloud integration platforms

2.2 Why is Cloud Integration Different?

Traditional integration approaches have been customized for on-premise setups, dealing with the security from intra-organizational perspective. The transition to the cloud increased the attack surface and introduced new challenges to the overall security. In the cloud, and especially when applying the integration scenarios among different actors, the interactions get executed among diverse, often unrelated entities. These entities exist in various organizations and often operate in different jurisdictions. The security of these flows needs to be properly assured and enforced with new approaches that acknowledge the heterogeneity and complexity of the whole environment.

2.3 Paper Outline

In the following section, we introduce *acUMA*, our extension that enables the use of novel UMA protocol in integration flows. We then analyze and discuss two approaches to cloud-based integration using standard OAuth 2.0 setup and UMA protocol. Finally, we assess the security of both protocols and provide a conclusion in the final section.

3. CLOUD INTEGRATION WITH ACUMA

For the purpose of evaluation of UMA-based integration flows, we have assessed available integration platforms and frameworks, focusing primarily on the possibility to support OAuth 2.0 and UMA protocols. We could not find any framework that supports UMA protocol. In our opinion, the primary reason for missing support for UMA is relatively recent stabilization of its core profile [16]. As the protocol is missing broad adoption among service providers, the incentive to support it in integration platforms and frameworks is still low.

In order to evaluate UMA flows in the domain of integration platforms we have decided to extend one of the existing frameworks and analyze protocol flows using reference platforms and demonstration prototype. Our primary selection

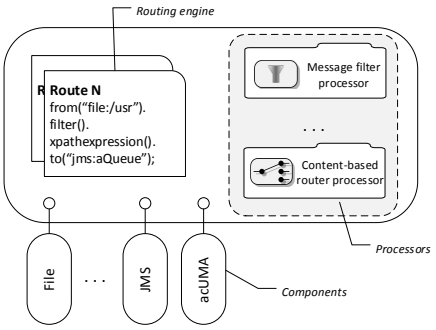

Figure 2: Architecture of Apache Camel

criteria for integration framework for extension was its openness, broad adoption, active community and good documentation. Based on these parameters, the *Apache Camel 2.15* has been selected. For integration with the web and API authorization framework, as well as for evaluation of OAuth 2.0 flows, we have adjusted and deployed *oxAuth* server [8].

Apache Camel is a Java-based framework, available under Apache 2 license. It can be deployed as a standalone Java application or integrated into other frameworks that provide the complete functionality of an *enterprise service bus* (ESB), such as Apache ServiceMix. Other projects that integrate Apache Camel include JBoss Fuse, OpenESB and Talend suite.

The architecture of Apache Camel is presented in Fig. 2, depicting three main fundaments of the framework and positioning our *acUMA* component among them. The core of this framework consists of a *routing and mediation engine* that implements *Enterprise Integration Patterns* (EIP) [10]. Based on the specification of routes, which can be done using one of Camel's *domain specific languages* (DSL), the engine manages the traversal and processing of messages in the system. This processing is performed by processors, which are also used to implement supported EIP patterns. Furthermore, the connectivity of framework with other systems is established through the *components*. They expose the platform, acting as a neutral interface between the systems. The part of the component that interfaces with other systems is abstracted as an *endpoint*, a factory that creates *producers* and *consumers* that interact with other systems in both incoming and outgoing directions, respectively.

In order to enable Apache Camel to take part in UMA-based flows, we have extended its functionality by implementing the *acUMA* component that enables consumption of UMA-protected endpoints. This extension is implemented as a reference prototype, consisting of the components listed in Table 1, along with their role. We have additionally adopted the existing *gauth* component and applied it to further work for the purpose of evaluation.

In our setup we have deployed one Apache Camel instance in Apache Karaf container, installed in a virtual machine on a VirtualBox host with i5-4300U CPU and 8GB RAM. This guest has been assigned with one vCPU and 1.5GB RAM. Additionally, we have deployed three more instances with the same configuration, taking the role of cloud application (based on Redmine), authorization server and resource server (based on oxAdmin). This setup has been used to support our analysis and provide practical insight into flows.

Table 1: acUMA Components

acUMARSController	manages communication with resource server
acUMAASController	manages communication with authorization server
acUMAProcessor	supports additional claim -based flows in UMA
refService	prototypical service functionality the application
dbLayer	persistence layer for application

4. ANALYZING INTEGRATION FLOWS

In this section we present the results of the analysis of integration flows performed in the test deployment environment, as described in the previous section. The deployment scenario assumes that the integration flows are executed between systems that belong to different organizational entities, located in separate domains and hosted completely in the cloud. We discuss and compare these approaches and their effects to *data security* and *user privacy* in the context of *integrated cross-domain services*.

4.1 The Scenario and Context

We analyze data flows among cloud services based on two protocols. The first one, governed by OAuth 2.0 [9], represents the common approach that is used by a majority of cloud services and applications to provide authorization consents for cross-entity data accesses. The second approach considers recent standardization efforts done by Kantara Initiative and IETF, based on UMA protocol [16].

In the following subsections we provide more details on each protocol and its application in cloud integration flows. In our analysis, we focus on the steps that are of interest for trust establishment and security in inter-entity interactions. The first activity, *authorization of a client*, enables a client to consume cloud resources on behalf of a resource owner. The second activity, *consuming cloud resources*, refers to the processing activity of an authorized client, repeated in separate and independent flows. This activity is also referred as *resource retrieval*, additionally covering the case of basic access to the resource. The client in our scenario refers to *integration platform* that consumes and processes resources of different *cloud services* on behalf of an *resource owner*.

4.2 OAuth 2.0

OAuth 2.0 builds on the concepts introduced by OAuth in 2010, with the primary aim to enable clients to access server resources on behalf of a resource owner [9]. Standardized in 2012, OAuth 2.0 has refined initial concepts and architecture of OAuth by introducing new types of flows and support for non-browser based applications, reducing complexity for client implementations and defining new token types.

In OAuth 2.0 scenario, a *resource owner*, an entity capable of giving access to *protected resources*, authorizes a *client*

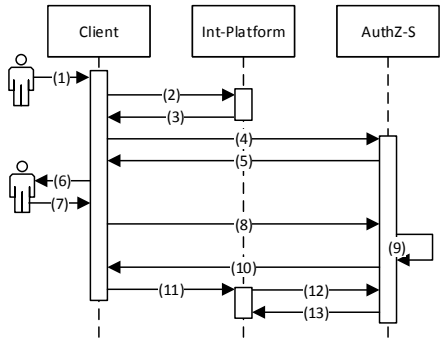

Figure 3: Authorizing int. platform in OAuth 2.0

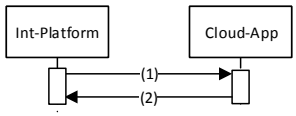

Figure 4: Resource retrieval in OAuth 2

to make requests to protected resources on behalf of a resource owner. A client is a device-agnostic term that refers to an application that accesses the protected resource hosted by *resource server*. *Authorization server*, the fourth entity in this setting, issues access tokens to the client, providing that the authorization consent by the resource owner has been previously obtained. Additional details on OAuth 2.0 specification and its application can be found in [9].

4.2.1 Authorizing Integration Platform

As defined in OAuth 2.0 flows [9], this process assumes that the integration platform has been previously registered with the cloud service authorization server, and that the *authorization code* grant type has been applied in the flow.

As depicted in Fig. 3, the flow starts with (1), where the process gets initiated by the user that navigates its client to integration platform and its interface (2), which requests the access to the external resource provider. After the resource provider is specified, the integration platform redirects the user's client to authorization server that governs the access to resource provider (3, 4). The authorization server then provides the client with the authorization interface (5) that is presented to the user (6) to provide consent.

After the access to the resources gets consented by the user (7), the client forwards the consent related data back to the authorization server (8), which processes the request and creates an authorization code (9). This code is then provided to the client in the form of redirect response (10), which gets forwarded to integration platform (11). The platform needs to present an authorization code to the authorization server (12) in order to get *access token* (13).

4.2.2 Consuming Cloud Resources

Access token represents the credential applied when accessing protected resources. It is provided in the form of a string and has a particular scope and validity. Using an access token, the integration platform can access resource server (1) and obtain the requested resource (2), as shown in Fig. 4. The process described in this figure is subsequently repeated, until the access token becomes invalid or expires.

As its validity is in practice set for a longer period of time, the possession of an access token enables the integration platform to access the resource in future independent sessions, respectively. Alternatively, OAuth 2.0 specification establishes an optional *refresh token* type which might be provided by authorization server along with the access token. Due to the similarity of flows we describe the process containing the access token and refer to specifications for further details [9].

4.3 UMA

UMA (User-managed Access) is OAuth 2.0 based protocol, developed as a result of efforts of User-Managed Access Work Group at Kantara Initiative and previous work based on User-Managed Access Control [14]. It is realized as a profile of OAuth 2.0, defining the flows, processes and APIs that govern the access to protected resources in distributed environment based on a resource owner policies and centralized authorization server. UMA specifications have been recently adopted as Kantara Initiative Recommendations and have been submitted to IETF. Core specifications of UMA include User-Managed Access Profile of OAuth 2.0 [16] and OAuth 2.0 Resource Set Registration [15].

UMA profile considers four entities to take place in interaction, similarly as in OAuth 2.0, and introduces *requesting party* as an additional entity that uses a *client* to access a protected resource. During the permission-gathering flow to access the resource, an authorization server might request from this party to authenticate or provide additional information. This process is governed in the scope of *claims-gathering flow*, another flow introduced by the specification. UMA defines two additional OAuth 2.0 protected APIs that govern access to authorization server separately for clients and resource servers. It also specifies two separate token types to be used by clients when accessing resource server (RPT) and authorization server (AAT). Additional details are provided in specifications [16].

4.3.1 Authorizing Integration Platform

In contrast to OAuth 2, UMA does not require user's presence in this step. In its standard flow, as shown in Fig. 5, UMA assumes that the integration platform, as a *client*, tries to access the resource without providing an *access token*, referred as *RPT* (1). This results in resource server to dynamically request the access decision from authorization server (2). Considering that the client (integration platform) is unknown, the authorization server denies the request (3). The resource server then registers necessary permissions for a particular client, access and resource and requests a *permission ticket* (4). This ticket (5) enables a resource server to instruct the integration platform with further required steps (6). The platform then uses the ticket to request authorization data from authorization server (7). Along with the request, the platform provides its *AAT* token, obtained during previous registration at the authorization server.

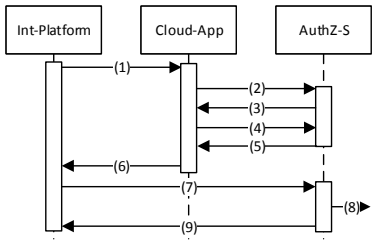

Figure 5: Authorizing cloud application in UMA

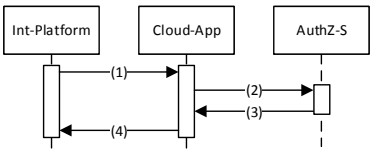

Figure 6: Resource retrieval in UMA

Depending on client's status and authorization policy for a particular resource, the authorization server might require additional claims to be provided (8). This process can be performed out-of-the band, in automatized manner, or manually, by involving a requesting party. In the next step, the authorization server provides a client with an *RPT* token that is used to access the resource under particular setting (9). This token enables the client to access the resource. It is however subjected to verification during every access, enabling dynamic evaluation of authorization policies.

4.3.2 Consuming Cloud Resources

As it can be observed from Fig. 6, when accessing external resources using standard *bearer token profile* [16], UMA assumes that each access should be separately evaluated and checked. By integrating this requirement in the standardized flow, UMA approach enables the separation of resource and authorization server, enabling their operation in different domains and cross-organizational setting.

Upon getting access request (1) that contains an RPT token, the resource server verifies its validity with the authorization server (2). Only when the permission is still valid, subjected to dynamic evaluation of access policies, the authorization server will grant the access to the resource (3) and resource server will subsequently provide that resource to the integration platform (4).

4.4 Comparing OAuth 2.0 and UMA

In this section we compare the flows of both protocols and consider the capabilities that affect the security of interactions. Although they belong to the same family of protocols, we can observe that OAuth 2.0 and UMA provide different levels of support for security and flexibility in the flows.

4.4.1 Authorization of Peers

In OAuth 2.0, client authorization is performed with the assistance of resource owner, which grants the *access consent* under particular *scope*. This can be characterized as a *synchronous* process, as the flow gets blocked until the resource owner consents the access. It can be also described as an *explicit* authorization, as the consent is provided explicitly.

In UMA, client authorization is performed out-of-the band, using separate flow that is not tight to access request. Therefore, a resource owner does not need to consume the services of cloud app or take part in its flow in order to introduce it to an authorization server. Instead, the resource owner defines authorization policies on authorization server, setting the particular conditions and requirements that apply to a specific client, or a family of clients. Based on a separate flows and policy-based authorization, this process can be characterized as *asynchronous*, based on in *implicit* consent.

4.4.2 Distributed Access Control

As Fig. 4 suggests, the authorization server is not included in the evaluation of OAuth 2.0 access requests. This is the result of a neutral specification that does not prescribe the methods and flows for evaluation of access requests. The process of coordination is therefore left to particular implementations. The first consequence of that is the *coupling* of resource and authorization servers, practically implying that they need to reside in the same organization or site in order to effectively coordinate token validation. The inclusion of authorization server might not be necessary at all, as the same effect can be accomplished by relying on shared database system used by both instances, or on some other component.

The second consequence of this non-specification is that resource servers are free to interpret possible errors or negative results obtained in the process of validation. The specification does not deal with the details and granularity of error responses provided to the clients as well. This limits the scope of alternate flows in cross-domain execution environments, implying their reduction to a binary decision in the standard case, affecting the *dependability* of distributed systems [1], crucial for their interconnection.

UMA approaches this issue by specifying the *protection API* and *resource set registration* between resource and authorization server [15]. This allows the structured evaluation of access requests, enabling the decoupling of access and resource server and their deployment in separate, cross-domain context. The distributed access control, therefore, enables the deployment of one centralized server that will control the access to user's resources in different domains.

4.4.3 Evaluation of Access Requests

In Section 4.4.2 we pointed to the issue of coupling between servers resulting from omitted specification in OAuth 2.0. Contrary to that, UMA establishes the process of access request evaluation that governs the responsibilities both of resource and authorization servers. This enables detaching of authorization server from the infrastructure of resource provider and its relocation to resource owner's premises, or to a third-party domain.

It should be furthermore noted that one resource server can use different authorization servers, and that a particular authorization server can govern the accesses for different resource servers, located in various domains as well. This

opens the possibilities to run authorization server independently and provide its functionality in the form of a separate cloud service.

4.4.4 Authorization Scope and Granularity

The other distinction that characterizes the evaluation of access requests in OAuth 2.0 and UMA is the granularity of authorizations. OAuth 2.0 provides the access on a basis of a *scope*, which represents an abstract construct that refers to a resource-server specific environment. The authorization is not tight to particular resource or access request, but to hard-coded scope that implicitly encompasses the range of resources available to the client. The common application of a scope considers it as a functionality necessary to accomplish an action, such as *permission to read files*, *ability to submit videos*, or *obtaining user's personal information.*

In UMA, an access token is tight to a particular client and refers to one or more permissions. Each permission describes an individual resource set and client's entitlements over it. The granularity of access control is determined by abstracting the scope of a resource set. Hence, a resource server can internally consider a group of a resources, such as files located under particular directory, to belong to a particular set, or it can refer each file as a separate resource set. Such referencing enables more structured representation and evaluation of access scopes in distributed environments.

4.4.5 Authorization Policies

Authorization policies enable users to define a structured and reusable set of rules and requirements that are applied in the process of *policy enforcement.* The separation of definition and enforcement of authorization policies furthermore allows for a greater degree of flexibility, traceability and manageability, especially in complex and distributed environments. Authorization policies are considered as enablers of effective data protection and access control [5, 22].

The capability to support separate policy definition differs significantly between both protocols. While UMA enables users to define complex authorization policies and delegate their evaluation to an authorization server, the approach used in OAuth 2.0 does not foresee any form of policies. Instead, OAuth 2.0 model requires the involvement of resource owner in consent-gathering flow, establishing the explicit, consent-based delegation of access that does not consider any other form of information or requirement to be evaluated.

The form and scope of policies that are included in UMA, however, do not follow standardized or structured approach. While its architecture based on separate *policy evaluation* and *policy enforcement* entities purposely resembles XACML approach [20], UMA avoids the specification of policies and leaves this segment to be application and authorization server specific. Such approach enables faster adoption of architecture, but hinders the interoperability and portability of policies between different implementations.

4.4.6 Handling of Invalid Access Tokens

Similarly as the process of peer authorization, the handling of invalid or expired access tokens in OAuth 2.0 requires the involvement of the resource owner to gather a new access consent. Furthermore, due to the opaque property of standard access tokens both in OAuth 2.0 and UMA, the client system is not able to determine the reach of token validity. This results in an inability of OAuth 2.0 based flows to continue the flows if the resource owner is not present. In the case of UMA, when the token is invalidated or expired, the process might not require the involvement of a resource owner. This solely depends on the authorization policies set by the owner and dynamically evaluated on the authorization server. Furthermore, even when the intervention of a resource owner might be required, this interaction can be performed in out-of-the band process, independently of integration flow.

4.4.7 Gathering Claims

The additional feature of UMA that is not existing in OAuth 2.0 includes the capability to request initiation of *claims-gathering flows.* This process can be initiated when integration platform accesses the authorization server for the purpose of obtaining an RPT. Before this token gets issued to the client, authorization server can initiate this additional flow and request additional information to be provided. This information includes the authentication token, but might be extended with other claim types as well. Upon authenticating with the authentication server, the flow continues and the client can access the protected resources again.

This control has potential to involve the application of additional functions to improve the security of integrations, such as communicating service agreements, obligations [20] or providing other legally-related consents.

5. SECURITY ASSESMENT

In this section we approach both the current scenario based on OAuth 2.0 and the scenario enabled by our *acUMA* contribution. Considering the results and discussion provided in the previous section, we analyze the security of both approaches. Our assessment focuses on particular scenarios with service compositions based on integration workflows that access data and execute processes in the cloud, crossing heterogeneous domains on behalf of a customer. For the purpose of this security assessment, we adopted security goals defined in RMIAS framework [3], applying them to the context of integration of distributed cloud applications [25].

5.1 Confidentiality

In this category, the system should ensure that only authorized users can access the protected information. Considering the flows of both protocols presented in sections 4.2.1 and 4.2.1, as well the discussion related to authorization in sections 4.4.1 and 4.4.3, we can conclude that both protocols support the confidentiality requirement.

Be refining the confidentiality requirement with *the principle of least privilege*, the difference in conformance among both protocols could be observed. Introduced by Saltzer and Schroeder [21], and later discussed by Schneider [23], the principle of least privilege assumes the process execution to be based on a minimal amount of privileges needed to complete a task. Suboptimal support for this principle

can be observed especially in the case of OAuth 2.0, as it provides widely scoped access to third parties.

In Section 4.4.4 we have shown how both protocols dealt with the authorization and demonstrated improved granularity level present in UMA. However, even the UMA approach does not fully satisfy the principle of least privilege, as it does not catch all underlying contextual properties of an access request. Such properties include the information that describes the scope and a purpose of an access request, as well as agent's position in a cloud-based integration workflow. The enhancement of access control process with additional properties would allow to introduce the dimension of *contextual awareness*, enabling structured and denser characterization of access requests.

5.2 Integrity

While the dimension of confidentiality deals with the access to data, the integrity in the scope of web authorization protocols considers the assurance of data integrity and the absence of unauthorized modifications.

The data modification in previously described flows is based on client's capability to alter data on a resource server. Similarly to confidentiality, this capability is determined by the scope of client's authorization. Considering that OAuth 2.0 relies on a definition of wide access scope, as shown in Section 4.4.4, the means to constraint client's capabilities can be considered as comparably weak. The capability to rely on richer semantics and finer granularity for user's permissions enables UMA to restrict clients to individual access methods, on a level of particular resources, enabling the authorization servers to enforce more restrictive access control.

5.3 Accountability

In RMIAS framework [3], accountability refers to the capability of the system to hold users responsible for their actions. The supplementing work [7] defines the accountability as a concept that allows the monitoring of the use of information and the transparency of the whole process, so that the misuse of information could be determined, and misbehaving parties held accountable. Both web authorization protocols in their current setups do not consider accountability. This presents a challenge for a future work.

5.4 Non-repudiation

Non-repudiation refers to the ability of a system to prove occurrence or non-occurrence of an event, as well as participation or non-participation of a party in a transaction. Non-repudiation should be considered in juristic context, as it requires legally-valid and acceptable supporting proofs.

Support for non-repudiation was not of primary concern in the design of both protocols. As one of supporting blocks of non-repudiation consists of authentication, the omission of cryptographic functions in OAuth 2.0 renders it practically non-applicable for such requirements. Although not directly targeting non-repudiation, UMA claim-gathering flow introduced in Section 4.4.7 the means to authenticate clients and requesting parties using broadly adopted protocols [16]. These protocols include SAML 2.0 and OpenID Connect, standardized approaches used for the purposes of critical business-to-business and eGovernment transactions. Stronger authentication, however, does not grant complete support for non-repudiation [13], but provides one step further to securing cross-entity transactions.

5.5 Auditability

Auditable systems should ensure persistent and reliable monitoring of all actions performed within the system. The auditability of distributed systems is considered as a non-trivial requirement and has been already investigated in various domains [6]. The both protocols that are subject of this work do not explicitly consider audit capabilities in their design. Therefore, we consider the audit measures to be out of the scope of these approaches, delegated to be managed on other system layers or approached as a future work.

5.6 Authenticity and Trustworthiness

The requirement of authenticity enables actors to verify the identity and establish the trust in a third-party involved in the transaction. The integration of proper authentication methods supports the fulfillment of other security requirements as well, including the categories such as *non-repudiation*, *auditability*, *confidentiality* and *integrity*.

Verification of the identity in OAuth 2.0 encompasses the resource owner and client that access authorization server. In the case of *implicit authorization grant* [9], the verification is restricted to the resource owner only. In UMA-based flows, the verification extends to *requesting party*, an entity that employs a *client* (agent) to access the resources, and a *resource server*, an entity that registers resources and validates client access token at authorization server.

OAuth 2.0 does not explicitly prescribe the methods used for client authentication, leaving the specifics to be agreed between client and authorization server bilaterally. The authentication should be considered separately of identification, a process that establishes the identity of a third party, which is also not included in the scope of specifications and as such considered as out-of-the-band process. In UMA, similarly, the authentication is specified in the terms of OAuth 2.0 flows by default. The specification provides *claim-gathering flow*, as described in Section 4.4.7, enabling the clients to authenticate using secure and reliable protocols, such as SAML 2.0 and OpenID Connect. This enables authentication based on high-assurance frameworks, as well.

In cases of both protocols, clients are required to authenticate servers by the means of TLS-based authentication, with the further possibility to rely on extended validation, providing that the necessarily chained trust relations are given, and security related consequences considered [11].

5.7 Privacy

In RMIAS framework, privacy goal is supported by obeying privacy legislation and enabling individuals to control, where feasible, their personal information [3]. Translated to cloud-based integrations where data is exchanged across different entities, the accomplishment of privacy goal can be considered as a challenging activity. While OAuth 2.0 do not consider privacy in its design, UMA framework deals with the privacy by providing the hints and recommenda-

tions to implementators and operators in specification suite [16]. Although the distributed architecture of UMA enables a higher degree of privacy and establishes controls for its improvement, this topic can be still considered as a challenge for future work.

6. CONCLUSION

In contrast to other, on-premise based integration systems, the workflows in the cloud integration platforms aim to be dominantly executed in the cloud. Such setup includes not only the execution of the primary business processes on third-party cloud premises but the interactions with the other cloud and on-premise systems on behalf of a client, as well. In the course of these interactions, especially when integrating with public services and APIs, the flows specified in web authorization protocols are applied for the purpose of access control and resource sharing. The resulting transition of business process execution to the third-party environment exhibits complex setups, uncontrollable permissions, and unclear responsibilities of involved parties. Such application requires careful reconsideration of security requirements and the issues arising from the use of standardized approaches, not completely suited for complex interactions occurring in the cross-domain environment.

In this work, we have introduced the extension that enables Apache Camel integration framework to apply UMA protocol processes in its workflow. In our analysis of security related controls and functionality of both OAuth 2.0 and UMA protocols, we have discussed differences between these approaches and how they affect security in the cloud-based integration environment. We have demonstrated the better alignment of UMA with security requirements for cloud-based integrations. However, both protocols expose deficiencies in the overall security perspective.

7. ACKNOWLEDGEMENT

This work has been supported by the European Commission's H2020 Programme under the SUNFISH project (N.644666).

8. REFERENCES

[1] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1), 2004.

[2] F. Baude, I. Filali, F. Huet, V. Legrand, E. Mathias, P. Merle, C. Ruz, R. Krummenacher, E. Simperl, C. Hammerling, et al. Esb federation for large-scale soa. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 2459–2466. ACM, 2010.

[3] Y. Cherdantseva and J. Hilton. A reference model of information assurance & security. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, p. 546–555. IEEE, 2013.

[4] C. Coles and J. Yeoh. Cloud adoption practices & priorities survey report. *Cloud Security Alliance*, 2015.

[5] C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization policies. In *Programming Languages and Systems*, p. 141–156. Springer, 2005.

[6] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *University of California, Berkeley*, 2009.

[7] R. Gajanayake, R. Iannella, and T. Sahama. Sharing with care: An information accountability perspective. *Internet Computing, IEEE*, 15(4):31–38, 2011.

[8] Gluu Inc. GluuFederation - oxAuth, 2015.

[9] D. Hardt. The OAuth 2.0 authorization framework. 2012.

[10] G. Hohpe and B. Woolf. Enterprise integration patterns. In *9th Conference on Pattern Language of Programs*, pages 1–9, 2002.

[11] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *Financial Cryptography and Data Security*. Springer, 2007.

[12] M. Kleeberg, C. Zirpins, and H. Kirchner. Information systems integration in the cloud: Scenarios, challenges and technology trends. In *Future Business Software*, pages 39–54. Springer, 2014.

[13] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of fair non-repudiation protocols. *Computer communications*, 25(17):1606–1621, 2002.

[14] M. P. Machulak and A. Van Moorsel. Architecture and protocol for user-controlled access management in web 2.0 applications. In *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, pages 62–71. IEEE, 2010.

[15] E. Maler, D. Catalano, M. Machulak, and T. Hardjono. OAuth 2.0 Resource Set Registration. 2015.

[16] E. Maler, D. Catalano, M. Machulak, and T. Hardjono. User-Managed Access (UMA) Profile of OAuth 2.0. 2015.

[17] M. Pezzini and B. Lheureux. Integration platform as a service: moving integration to the cloud. *Gartner*, 2011.

[18] M. Potočnik and M. B. Juric. Integration of SaaS using IPaaS. In *The 1st International Conference on CLoud Assisted ServiceS*, page 35, 2012.

[19] P. Raj. Enriching the integration as a service paradigm for the cloud era. *Cloud computing–principles and paradigms. Wiley, Hoboken*, pages 57–96, 2011.

[20] E. Rissanen et al. extensible access control markup language (xacml) version 3.0, 2013.

[21] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[22] P. Samarati and S. D. C. Di Vimercati. Access control: Policies, models, and mechanisms. *Lecture notes in computer science*, pages 137–196, 2001.

[23] F. B. Schneider. Least privilege and more. In *Computer Systems*, pages 253–258. Springer, 2004.

[24] L. D. Xu. Enterprise systems: state-of-the-art and future trends. *Industrial Informatics, IEEE Transactions on*, 7(4):630–640, 2011.

[25] D. Zissis and D. Lekkas. Addressing cloud computing security issues. *Future Generation computer systems*, 28(3):583–592, 2012.