Structuring the Scope: Enabling Adaptive and Multilateral Authorization Management

Bojan Suzic*, Andreas Reiter† and Alexander Marsalek‡
Institute for Applied Information Processing and Communications
Graz University of Technology, Austria
Email: *bojan.suzic@iaik.tugraz.at, †andreas.reiter@iaik.tugraz.at, ‡alexander.marsalek@iaik.tugraz.at

***Abstract*—In this work, we examine an *access scope*, a concept in authorization management broadly applied for the specification of access constraints in web service integrations. By analyzing a typical use-case of cross-organizational cloud service automation, we show the suboptimal capabilities of static, coarse-grained and inflexible scopes that negatively impact security and management of service integrations on a web scale. Using the graph-based structure that relies on semantic technologies we introduce dereferenceable and self-descriptive authorization extents that allow expressive, granular and dynamic specification of security requirements. Through its application in the running scenario, we show how this construct can be administered to support confidentiality, integrity and privacy requirements of service integrations by allowing selective information sharing based on contextual properties.**

I. Introduction

The increasing degree of cross-system dependence and platform diversity present on the internet are transforming the security management of cloud service integrations towards progressively complex and costly endeavor. This is especially notable when considering emerging types of cloud services that rely on resource sharing and processing across multiple organizations to provide their users with advanced service automations or compositions [1]. Existing approaches broadly applied to manage the security of such integrations can satisfy the users' expectations and emerging legislative regulations only partially. This is due to their primary intent to establish the framework for authorization protocol flows [2, 3], while the aspects important for capability and expressiveness of security controls were left to be independently decided in implementations.

In the current ecosystem, prevailingly based on OAuth 2 framework [2, 4], the users are impeded from effectively controlling and managing the sharing and use of their resources by third parties. This issue is reflected from several perspectives. First, the *authorization scopes* employed in OAuth 2 are defined *unilateraly* by service providers and imposed on users and accessing clients. They are *static* during the application lifecycle, *coupled* to the service data models and business processes and *incompatible* among providers. Their design may reflect a *commercial goal* of service providers that often contradict with the security and privacy of user data. Finally, users lack mechanisms to *manage the security* of their resources hosted at different providers in an automated manner that is *consistent*, *context-sensitive*, *expressive* and *reusable* in heterogeneous environments. For additional details on these issues we refer to case studies [5, 6].

978-1-5386-0683-4/17/$31.00 © 2017 IEEE

Our work focuses on advancing the security of complex interactions performed in the scope of Web APIs, which represent a common building block of inter-organizational collaborations in the cloud [4, 7]. For this purpose, we address the challenge of service-coupled, implementation-specific and unilaterally established security management controls. We detach the security management from particular interfaces, allowing this process to be performed using flexible and multilaterally adjustable controls that are consistent and machine-understandable across the environments. By relying on graph-based representations, we introduce structured and self-descriptive authorization extents that support user-centric specification of security controls without imposing significant integration requirements for service providers.

Paper organization. In Chapter II we provide the brief overview of authorization management in service-integrations. Considering OAuth 2.0 web authorization framework and the typical use case present in many scenarios, we show the security issues of underlying techniques and derive the requirements for integrated and cooperative cross-organizational authorization management. Chapter III introduces DASP framework, which serves as a basis for our work, and proposes our specification model based on management flows and authorization descriptor. This contribution is further elaborated in the subsequent chapter and examined using a running case scenario. Chapter V evaluates and discusses our work, followed by the overview of the related work and the conclusion.

II. Cross-System Authorization Management

A. Web Authorization and Service Integration

The approach of cloud-based integration got broader attention recently, as the products focused on integration and management of cloud services started to 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 [8]. In the focus of our work are data exchanges performed on the web, using HTTP protocol and RESTful interfaces, which currently represents one of the major approaches for Web API implementations [4].

Currently, OAuth 2.0 [2] is broadly adopted mechanism for the protection of Web APIs [4]. Building on it, UMA [3] represents an emerging protocol that further refines its

TABLE I: Overview of requested and necessary permissions across integrated services

Srv.	Req. scope	Scope description (declared by service)	Actually necessary data or activities
(1)	*gmail.compose*	Create, read, update and delete drafts. Send messages and drafts.	None.
(1)	*gmail.modify*	All read-write operations except immediate, permanent deletion of threads and messages, bypassing trash.	Retrieve message sender extracted using *From:* header field of a recently received message.
(2)	-	All operations exposed by the Web API.	Add entity to a particular list of subscribers.

flows, processes and APIs, focusing on the user-centric protection of resources in distributed environments. Another approach that deals with authorization, considering distributed and enterprise-oriented perspective, is XACML [9], an XML-based declarative language standardized by OASIS. XACML provides the means to specify access control policies based on an extensive set of built-in data types, functions, combining algorithms and profiles. However, due to its focus on a single enterprise and the overall complexity [10], the practical adoption of XACML in multi-organizational environments is restricted. In addition, XACML lacks the mechanisms to request permissions and establish authorizations across systems, which is a crucial requirement for modern service integrations. In this segment *consent-based* OAuth 2.0 [2] with its simple structure and authorization flows is widely applied [4].

B. Motivational scenario

To illustrate the problem and the application of our framework we describe the following running scenario, which relies on services known as *cloud integration platforms* [8, 11]. Consider an organization *MyOrg Inc.* that uses the service of an external platform, in this case *Zapier*[1], which acts as an agent that integrates several data sources and services provided by other companies on behalf of its customer. Precisely, *Zapier* accesses *Gmail* and *MailChimp*, to retrieve and process data on behalf of *MyOrg Inc. Zapier* periodically connects to *Gmail* to get recent emails. After retrieving emails, it extracts email senders and adds them as subscribers to MailChimp marketing platform. This step is repeated periodically, typically in the range of 5-30 minutes, whereas only the recently received messages are relevant, and only a particular list of subscribers at MailChimp has to be updated.

To obtain necessary permissions, in its setup process Zapier creates permission request using OAuth 2.0 authorization flow [2]. These permissions have to be consented by the user, which allows the service provider to issue long-lived access token to Zapier. Fig. 1 shows the related consent interfaces at Gmail (left) and MailChimp (right). Note that Gmail explicitly asks the user to authorize the access. As MailChimp does not compartmentalize its API permissions, the user is not asked for explicit consent. Instead, by logging in, the user implicitly agrees to provide the full API access to Zapier on its behalf. This is typically described as *connecting accounts* at integration platforms.

Table I provides an overview of requested authorization scopes with their coverages declared by the services. The last column describes the permissions that are actually necessary for the platform to successfully accomplish its integration task using GMail (1) and MailChimp (2).

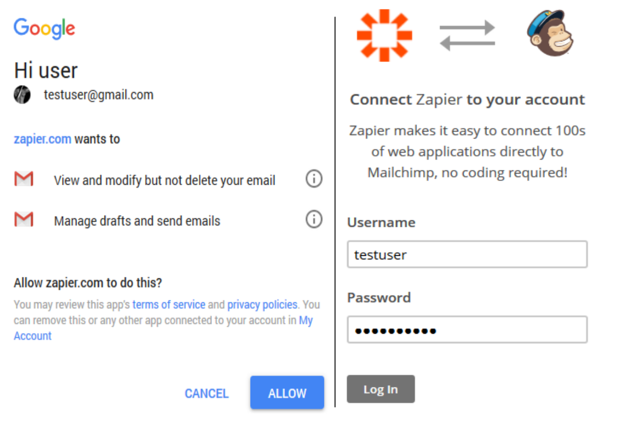
[1] Note that there are many similar commercial offerings. For a market overview we refer to [12].

From Table I we can observe the *overpermissioning* of the integration platform at both services, which does not obey *the principle of least privilege* [13]. The only action necessary at GMail is to retrieve the value of *From:* field from the header of recent messages. Zapier, however, asks and gets permissions to retrieve and manage all email messages at user's (or organizational) GMail account. It can also manage drafts, send or temporarily delete messages from the given account, although none of these activities are actually necessary for the successful task completion.

The extent of both of the provided scopes creates a potential for a range of vulnerabilities. If not acting honestly, Zapier can retrieve all messages, to profile organization or the user, or to gather and process private information of organizational customers. This part opens additional unmanageable risk for external parties, as the message senders may not have any idea that a third-party service is given the permission to read their emails. Being authorized to send the messages in the name of the account owner, Zapier can manipulate third-parties, alter or temporarily delete (hide) important conversations.

While the permissions given at GMail are somehow still restricted, Zapier is authorized to retrieve any data or execute any API operation on behalf of MyOrg Inc. at MailChimp. Such broad permissions may lead to the retrieval or altering of personal data of subscribers registered in user's account. Other activities, such as spamming or distributing malware to targeted users can be executed as well. This is due to the design and implementation of MailChimp API, which does not implement any further restrictions for API accesses.

In the above scenario, the integration platform is an external entity that accesses services using their exposed Web APIs, which is a broadly adopted technique to share data and consume services across domains. Web API implementations typically follow RESTful architecture and control client accesses using *API keys* [14] or *OAuth 2.0* [2].


Google

Hi user

testuser@gmail.com

zapier.com wants to

View and modify but not delete your email

Manage drafts and send emails

Allow zapier.com to do this?

You may review this app's terms of service and privacy policies. You can remove this or any other app connected to your account in My Account

CANCEL ALLOW

Connect Zapier to your account

Zapier makes it easy to connect 100s of web applications directly to Mailchimp, no coding required!

Username

testuser

Password

••••••••••

Log In

Fig. 1: Zapier: obtaining consents from users

OAuth 2.0 establishes *authorization scopes* as a mean to control the extent of the authorization. As each service is responsible for specification and enforcement of supported scopes in its domain, the capabilities of scopes often differ among services. Generally, due to coarse extents, they tend to provide broader permissions than required for the use case, which we show on Table I.
Note that this problem applies generally, beyond the presented use case, as many organizations use this technique to control resource sharing with external entities.

C. Inherent Characteristics of Access Scopes

Access scopes allow service providers to establish abstract ranges of coarse-grained permissions as referenceable concepts that can be reused across organizations to communicate and provide authorization consents. OAuth 2.0 specification introduces *scopes* as parameters with the purpose to (1) allow clients to specify the scope of their access requests, and to (2) enable providers to inform the clients about the range of accepted or provided permissions [2]. The permissions implicitly communicated within the OAuth 2.0 *scope* are associated with each access token provided to the clients for single or repetitive accesses [15].

OAuth 2.0 defines the structure of the scope parameter as a list of space-delimited, case-sensitive strings, whose inherent permission extent is controlled by the provider. Multiple of these strings (scopes) can be combined in scope parameter to express a combined range of access permissions. Besides recommending that service providers should document their scopes, OAuth 2.0 specification does not provide any additional details that would allow dereferencing of scopes or establishing of cross-system interoperability at a higher level, beyond the opaque strings and hard-wired logic [16].

Following the illustrative example presented in the previous section, in Table II we briefly summarize the important properties of access scopes that restrict their capability to precisely control security and privacy aspects. In addition to that, the properties (C1) and (C3-C5) hinder the scalability of the overall concept beyond the boundaries of a single system or an organization. This is due to the overhead imposed to support the creation and maintenance of point-to-point integrations in the otherwise massively point-to-multipoint interconnected environment.

The concept of access scopes, as currently embraced, allows authorized clients to perform unlimited accesses, restricted only by abstract, unilaterally defined and non-inferrable scope coverage. As scopes relate to concepts, rather than to data or object instances, their application to control access based on properties of protected resources is practically not feasible. We can observe this aspect from the example presented in Section II-B and Table I, where the scopes allow restricting accesses using only a range of predefined activities, without the possibility to determine a span of accessible resources. The similar limitation applies to the specification of dynamic data transformations, which are necessary to implement advanced privacy and legal requirements. Moreover, the overall mechanism does not accommodate the derivation of contextual confinements, which allow access control based on a range of contextual parameters. These limitations stem from the tight expressive capability, lack of scope structure and its relational detachment from systems and environment.

D. Requirements

Based on a typical use case and the identified issues, we derive requirements for integrated and cooperative cross-organizational authorization management as follows:

(R1) Authorization establishment: the actors residing in different domains should be enabled to establish cross-organizational authorizations for data exchange or resource consumption among domains.
(R2) Granular specification of accesses: users should be able to specify and update a customized view consisting of a range of resources or operations that satisfy their resource sharing requirements using fine-grained controls.
(R3) Claiming acceptable constraints: clients should be able to discover and claim acceptable constraints on resources, such as partial representation or transformation.
(R4) Context-dependent enforcement: resource owners should be able to specify dynamic authorizations that rely on the properties of a resource or environment.
(R5) Transformational interactions: resource owners should be able to specify online transformation of resources with the purpose of reducing exposure of sensitive and process-irrelevant data.
(R6) Scalable management: the administration of requests and authorizations should be detached from platforms and specific implementations, allowing the actors to apply reusable approaches under different systems.
(R7) Supporting autonomous agents: the framework should enable automated integration of agent-based actors, allowing autonomous discovery, establishment, and

TABLE II: Characteristics of scopes

N.	Property	Description
(C1)	Unilateral definition	Scopes are determined by the service provider, without considering the requirements of resource owners or clients. Only predefined set of scopes can be referenced by adjunct parties.
(C2)	Invariable	Determined statically, scopes represent immutable sets of coarse-grained permissions whose extents are not intended to be changed over time in production environment.
(C3)	Unstructured	Scopes are defined as opaque strings without decomposable data structure that could support discovery of their extents or enable the dynamic definition of new scopes and their properties.
(C4)	Out-of-the-band	The scope extent is communicated non-transparently to applications, preventing them from deriving, understanding or impacting of underlying properties.
(C5)	Coupled	Scopes cannot be decomposed or synthetized based on the permissions and concepts encompassed in them. Only predefined scopes with built-in properties are supported.
(C6)	Context insensitive	The contextual properties, such as environmental attributes or resource features cannot be expressed.

management of authorizations across the interaction chain.

III. Adaptive and Cooperative Authorization

In this chapter we define the flows that support cooperative and adaptive authorization management. We then introduce *authorization descriptor*, a graph-based structure that allows granular, instructive and flexible expression of authorization requirements and grants.

A. DASP framework

Our work is complementary to **DA**ta **S**haring and **P**rocessing (DASP) framework, whose goal is to establish building blocks for achieving externalized and decentralized authorization that spans across heterogeneous services and addresses the challenges related to the unified and dynamic management of security controls in connected environments. The framework itself consists of (1) the architectural and interaction model, (2) supporting semantic vocabularies and (3) tools that demonstrate its application and allow a broader integration.Additional details are available in [5, 17].

In its interactions DASP considers the following entities:

1) *Service provider (SP)* hosts resources, provides services or performs data processing;
2) *Resource owner (RO)* or *user* owns resources hosted at SP or subscribes to its exposed services;
3) *Client (C)* accesses resources or consumes services at SP on behalf of RO or according to its policies;

The overall architecture considers *provider-centric* and *user-centric* deployment models for authorization management, as shown in Fig. 2(a,b). Both models assume the externalization of authorization functionality to a separate component, responsible for operations such as the definition, evaluation, and enforcement of security goals. Shown on Fig. 2 as DG *(data security gateway)*, this component allows resource owners (RO) to specify a set of policies that are imposed on client accesses to the resources exposed by different service providers (SP). This approach aims to achieve consolidated and unified authorization management by relying on the common abstract set of concepts for the description of exposed resources (Web APIs) and security policies applied to protect them.

Using the common flows (M on Fig. 2), the users are able to employ a coherent and consistent set of tools and services to manage the security of their resources distributed at multiple providers [5, 17].

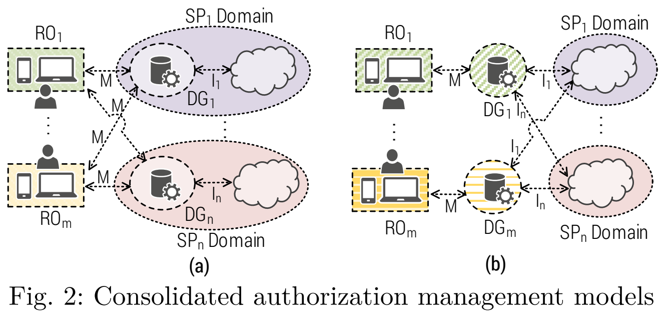


Fig. 2: Consolidated authorization management models

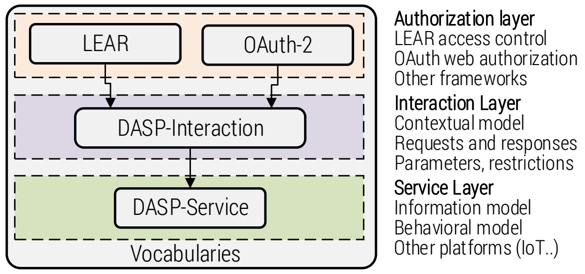


Fig. 3: DASP Framework: vocabularies and layers

B. Vocabularies

DASP framework envisages the modular definition and integration of a range of vocabularies using a bottom-up approach [18], enabling users to select the constituents that best fit their needs. Fig. 3 depicts an overview of descriptive layers and vocabularies provided in the framework. We model the domain using three layers. Service layer provides the view of information and behavioral model of a web service. In the second layer, we provide *DASP-Interaction* to support the descriptions of requests, responses, contextual properties and resource restrictions. The third layer is dedicated to vocabularies that allow the integration of different access control models and frameworks.

Prior work established *DASP-Service* and *LEAR*[2] vocabularies for the description of RESTful services and their integration with LEAR, an access control model for Web APIs [5]. In this work we refine this contribution with the second, intermediate layer. We also establish *DASP-Interaction* and *OAuth 2* vocabularies to support the modeling of authorizations using various frameworks. For further details we refer to our previous work [17] and publicly available[3] tools, vocabularies and reports [19].

C. Management flows

(S1) the service descriptor: based on available resources and their organization, the provider exposes service description that includes resource structure and applicable operations and constraints on exposed entities.

(S2) Determining the request scope: based on the resource model provided by the service, the client decides the extent of requested authorization. This is performed by finding an intersection between security and functional goals of the client, considering the exposed resources, applicable constraints and operations on the server side.

(S3) Requesting authorization: the client coins an authorization request, which can be delivered using two interaction types. *Interactive request* assumes the presence of resource owner in the authorization flow. *Asynchronous request* is performed as an independent flow, without the direct involvement of a resource owner.

(S4) Refining authorization extent: in this step the resource owner may inspect and refine the request. Depending on request flow, this can be done using the owner's client involved in the *interactive flow*, or at the side of the SP, after the *asynchronous flow* has been performed.

(S5) Transforming into security policy: once inspected and consented by the resource owner, the authorization request

[2] Lightweight and Expressive Access Control for RESTful services

can be server-side transformed into security policy using a format that is specific to the target system.

(S6) Inspecting authorization descriptor: after the security policy is derived and integrated into the target environment, the provider optionally delivers the authorization descriptor back to the requesting party. The purpose of this step is to support *cooperative authorization*. It also allows the clients to infer the status and extent of actions and operations finally allowed in the process.

D. Authorization descriptor

Consider the vocabulary Θ as a 4-tuple $\Theta=\{C, R, E, I\}$, where C, R, E, I are sets that respectively denote classes, relations, instances of classes and relations, and axioms. Note that classes are seen as unary predicates, while relationships denote binary and n-ary predicates. Axioms that are common in vocabularies are *subclass* and *subproperty*, which apply to the classes and relationships, respectively. Other axioms include *domain* and *range*, allowing to restrict the classes that can be used with relations.

In vocabulary Θ we consider the elements from sets C and R as *terminological knowledge*, referenced as *TBox*, and instances with their attributes as *assertions*, denoted as *ABox* [20]. Hence, a TBox model serves as a *shared conceptualization* [21] that enables interacting parties to derive the structure and information about APIs, authorizations and processes described with $E\in\Theta$. The vocabularies present in DASP layers (Section III-B) serve to provide such terminological knowledge, establishing domain- or application-specific means to map the meaning of the concepts between different systems and organizations.

The underlying design decision has been motivated by the issues C3-C5 (Table II), which impose a significant overhead to establish and maintain interoperability and transparency of controls between systems. As an example, consider the scope *gmail.compose* from Table I. Its meaning is communicated between the provider and the client using human-readable documentation. Both entities have to integrate this inherent meaning by means of hardwiring. For each service provider using such mechanism, the client has to implement service-specific scopes and maintain them across API versions. Our approach aims to achieve decomposability and machine-interpretability of such scopes by relating them to abstract concepts whose meaning is shared across many parties. In other words, it advances the interoperability from *syntactic* to *semantic* level [16]. This lowers integration obstacles for both providers and clients as they can reuse existing building blocks to derive knowledge and establish many-to-many mappings automatically. Use of semantic technologies is further motivated in [22].

The service semantics (S1 in Section III-C) is established by instantiating the concepts from *DASP-Service* as ABox-knowledge, and representing them using a language such as OWL [23]. By relying on the same basis, authorization descriptors apply a similar approach.

Having $AuthorizationRequest$, $AuthorizationResponse$ and $ErrorResponse$ defined under $OAuth{-}2$ vocabulary as subclasses of *Request* and *Response* in $DASP{-}Interaction$ vocabulary, we define authorization descriptor as a structure that instantiates classes and relationships from $DASP{-}Interaction \sqcup OAuth{-}2 \sqcup DASP{-}Service$ so that it contains at least one instance of $AuthorizationRequest$, $AuthorizationResponse$ or $ErrorResponse$. These instances then have to apply domain and range restrictions, as defined in vocabularies[3], and conform to service capabilities, as exposed in the model of a target service. They may reference only the instances that are exposed in the service model, including actions, resources, their elements and applicable restrictions.

Note that *authorization descriptor* may take different roles, such as the request to provide authorization or response that describes given permissions. In the case of the authorization request, the instance of *AuthorizationRequest* has to provide further details on requested authorizations, by referencing the actions (intents) or resources exposed by the service and claiming a range of acceptable restrictions and operations, as provided in the service description. The underlying TBox model is used to derive the capabilities and meanings of exposed entities using class and property descriptions and relationships.

To illustrate the practical application of this structure using the running case we refer to Fig. 4. Sec. IV provides additional details on the application of this structure with the regards to management flows defined in Sec. III-C.

IV. Design and Operation

In this chapter, we examine the application of our proposal considering a case scenario motivated in Section II-B. To support comparable functionality and the application in other scenarios and protocols, we then elaborate particular functional aspects of our proposal.

A. Application in the running case scenario

Our contribution reduces the over-permissioning shown in motivational scenario (Section II-B) both in *static* and *dynamic* dimensions. To illustrate this, we refer to Fig. 4, which depicts authorization descriptors as graph-based structures. Each node on the figure refers to a class instance from the respective vocabulary, whose type is shown as the acronym[4]. The colors of nodes indicate their definition vocabulary. Edges represent object properties (with the range of class instances) and data properties (with the range of data values). While their actual names are not relevant for the functionality, we have labeled class instances using free-text for the purpose of readability.

To retrieve the messages and their content, integration platform needs access to two actions:

1) *List emails*, which allows retrieving a list of emails consisting of message ids and thread ids.
2) *Retrieve email*, which uses message id and HTTP GET to retrieve the message content as JSON.

For this purpose, the accessing client creates two requests, shown in Fig. 4 as (b) and (a). In our assumption, the client is *partially cooperative*, requesting non-optimally restricted authorization scope, whose structure is differentiated as *client request* on the figure.

[3] Due to space constraints we point to http://daspsec.org for an overview of vocabularies and http://demo.a-sit.at/am for tools

[4] AR:AuthorizationRequest, EL:Element, A:Action, SE:SanitizeElement, ER:ElementRestriction, TD:Today

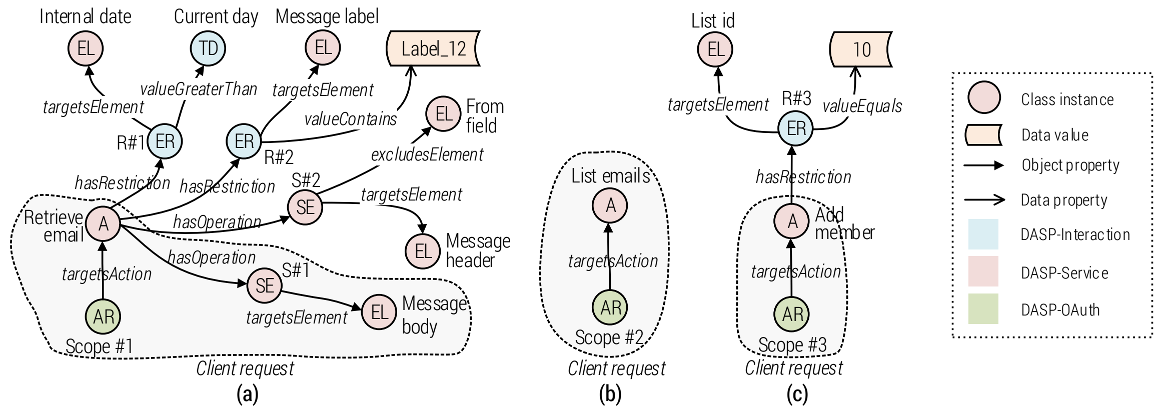


Fig. 4: Zapier: defining authorization scopes for GMail and MailChimp

In the next step, the client forwards the authorization request containing the requested scope to the service for further processing. This enables the resource owner to perform the *scope redaction* (S3-S4 in Section III-C), allowing it to adjust the scope to conform to its specific security and privacy goal. In our use case, prior to giving the authorization consent, the user augments the request with further restrictions and transformative operations, as shown in Fig. 4(a). This is done by examining exposed service model and altering the provided structure according to exposed resources and operations. Hence, knowing that the integration platform does not need all requested operations and parts of the resource, the user augments the scope as follows:

1) *R#1*: expose only messages whose internal date is restricted to the current day (transaction time)
2) *R#2*: expose only messages whose label is flagged with value *Label_12*
3) *S#2*: perform operation *Sanitize Element* on message header, so that all its consisting parts except *From field* get sanitized prior to the delivery

With this structure, the user practically reduces the range of available resources not only based on the type of operation or activity but also using the *contextual properties* of the resource (R#2) and *dynamic* feature such as the relative temporal position of the transaction (R#1). Based on these requirements, prior to the delivery, the message gets dynamically processed with the goal to prevent client seeing unnecessary data (S#2). In this sense, the integration platform is allowed to retrieve only the messages received on the day of the interaction, and from them, it sees unmasked only the data content of *From:* header. The structure of the message is preserved according to the data model of API.

Similarly, the authorization scope at MailChimp is defined by the platform and augmented by the user, as shown in Fig. 4(c). In this case, the user restricts the authorization to the action of adding subscribers using only the subscriber list whose *id* equals to *10.* The platform is not allowed to access any other functions of the API, nor to amend the subscriber lists other than this one.

B. Structuring service descriptors

To expose a service descriptor, the service provider first has to structure and provide its service model using a common framework. For this purpose we rely on DASP-Service vocabulary (Fig. 3 and Section III-B), which can be reused by service providers to describe information and behavior aspects of their services. Providing the abstract concepts, these vocabularies allow providers to flexibly determine and apply different levels of granularity and detail, as it suits their scenarios. The resulting descriptor may be simplified, presenting only resources on a high-level, or it can be enhanced to reflect the more complex structural, behavioral and authorization aspects. We illustrate the overall approach as follows.

Given the service vocabulary $\Theta_{(s)}=\{C,R,\varepsilon,I\}$, a service provider instantiates its service model $M=\{C_M,R_M,E_M,I_M\}$ so that $C_M\subseteq C$, $R_M\subseteq R$ and $I_M\subseteq I$ holds and $\forall e\in E_M, e\in C_M \vee e\in R_M$. Elements from set E_M denote instances of classes and relations defined in vocabulary $\Theta_{(s)}$. Note that, depending on complexity and description goals of the service provider, several vocabularies provided in the common framework can be used to establish the service model.

Several formats can be used to present these graphs, including RDF, JSON-LD or Turtle. To illustrate the overall process, we provide simplified and reduced version of Gmail descriptor using Turtle syntax in Fig. 5. The first part of the descriptor denoted as $\textcircled{1}$ establishes the prefixes and the base used in the document for the referencing of concepts from various vocabularies.

In $\textcircled{2}$ the instance of *service* class is initialized, which serves as a root node for described service. Following that node, we can identify the actions (intents) that are exposed by the service as well as the available and affected resources by each action. The action itself is determined by the sequence of (dynamic or static) URL path elements and the HTTP methods for API calls, as partially shown in the figure. *EmailResource* on the figure refers *MessageHeader* as one of the elements it provides. This element is further elaborated under $\textcircled{3}$, where its (sub)elements are described. The purpose of $\textcircled{4}$ is to express the extraction rules necessary to retrieve and evaluate the values of resources or their elements. This activity is often referenced as *semantic lifting* [24].

This example shows two cases, where the action or the nested element serve as data structure containers, respectively. Using such rules the systems can implement restrictions, such as *R#1 and R#2*, or perform transfor-

```
@prefix : <http://www.daspsec.org/o/gmail-api#> .
...
@base <http://www.daspsec.org/o/gmail-api> .                                   ①
<http://www.daspsec.org/o/gmail-api> rdf:type owl:Ontology ;
  owl:imports  <http://www.daspsec.org/o/dasp-service/0.3>, ... .
:GmailService rdf:type dasp-service:Service ;
    dasp-service:hasAction :RetrieveEmail .
:EmailResource rdf:type dasp-service:Resource ;
    dasp-service:hasElement :MessageHeader , ... .                             ②
:RetrieveEmail dasp-service:Action ;
    dasp-service:affectsResource :EmailResource ;
    dasp-service:hasURLPath:1 :U_URLDesignator .
    ...
:U_URLDesignator dasp-service:StaticPathElement ;
    dasp-service:hasElementValue "gmail"^^xsd:string .
:U_MessageIdAPIElement rdf:type dasp-service:DynamicPathElement .
:MessageHeader rdf:type dasp-service:Element ;
    dasp-service:hasElement :FromField , ... .                                 ③
:FromField rdf:type  dasp-service:Element .
:MessageHeaderExtractor rdf:type dasp-service:ElementExtractor> ;
    dasp-service:hasElement :MessageHeader ;
    dasp-service:isProducedByAction :RetrieveEmail ;
    dasp-service:hasJSONPathContentExtractionRule "$.payload.headers"^^xsd:string .
:FromFieldExtractor rdf:type dasp-service:ElementExtractor ;                   ④
    dasp-service:hasElement :FromField ;
    dasp-service:hasElementContainer :MessageHeader ;
    dasp-service:hasJSONPathContentExtractionRule> "$..[?(@.name==\"From\")]"^^xsd:string .
```

Fig. 5: Excerpt for Gmail service descriptor

mative operations such as *S#2*, as shown in Fig. 4.

C. Consuming service descriptors

The consumption of service descriptors is conceived as traversing the provided graph structures and integrating the results in the local environment, according to the role of classes, their hierarchical dependencies and applied object and data properties. The process of establishing authorization request from such structure can be described as follows. We apply the notation $x \xrightarrow{z} y$ to denote that the instance x connects to y using property z. Considered as a graph, x and y would be represented as nodes connected by directed edge z.

First the agent has to retrieve service descriptor D from the adjunct system. The exposed services are then derived as $S \leftarrow s_d \in D \mid s_d.instanceOf(DASP\text{-}Service{:}Service)$. After selecting the relevant service $s \in S$, the agent retrieves its exposed resources $R \leftarrow res \in D \mid s \xrightarrow{hasResource} res$ and actions $A \leftarrow act \in D \mid res \xrightarrow{hasAction} act$. The agent might want to interact with particular resource, or to execute some exposed action, with the latter being more specific. Using the later, the agent initializes a new scope sc with $sc \xrightarrow{targetsAction} act \mid act \in A$ and proceeds on with deciding which range of exposed operations are deemed acceptable for its use case. This is done first by finding a range of affected resources and their elements using $act \xrightarrow{affectsResource} res$ and $res \xrightarrow{hasElement} el$, and then inspecting a range of supported operations $op \in OP \mid (el \vee res) \xrightarrow{isSupportedBy} op$. The acceptable operations can be amended to the action with $act \xrightarrow{hasOperation} op$, optionally with parameters supported by the operation, as shown on Fig. 4(a). Depending on the client preferences, the resulting scope may include arbitrary levels of detail. In any case, the user may redact the scope prior to confirming the authorization.

D. Redacting and updating the authorization descriptor

The process of scope redaction (S4 in Section III-C) is performed similarly as structuring and consuming of service descriptors, as it relies on the same underlying entities. First, the application (local or remote) has to receive the authorization descriptor. Based on the vocabularies used in this descriptor (e.g. prefixes on Fig. 5), the application retrieves the respective vocabularies (TBox knowledge) and referenced service descriptions (ABox knowledge), learns their structures and checks the conformance of instances in authorization descriptor with them. Then, the application derives supported resources, actions or operations and renders the user interface to allow the user to customize the authorization descriptor with supported entities.

Note that the redaction can be performed repetitively, enabling the users to adjust their authorizations at the arbitrary point in time. We have implemented the server-side application that provides this functionality[3].

E. Integration with other frameworks

We aim to provide a protocol, language and system-agnostic contribution that is applicable beyond a single use-scenario or a platform. From this point, we describe its integration with OAuth 2.0 web authorization framework [2], which is practically the most broadly adopted approach to manage cross-domain authorizations [4].

Fig. 6 shows the typical protocol flow in OAuth 2.0 authorization code grant. The steps (0) and (2b) represent the additions to a standard flow that need to be implemented to support the proposed integration. These steps correspond to management flows (S1) and (S4) (Section III-C), respectively. The approach to execution of activities encompassed in (0) has been described in Section IV-B and Section IV-C. We envisage the implementation of the second step (2b) using client-based or server-based scope inspection. While the former type assumes the interception and redaction of the requested scope in the realm of the client-software[5], the latter assumes these activities to be performed using server-side interface.

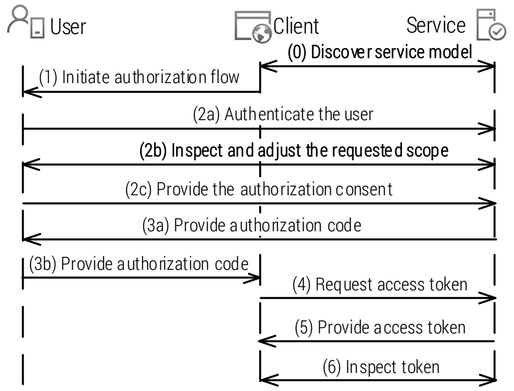

User
Client
Service
(0) Discover service model
(1) Initiate authorization flow
(2a) Authenticate the user
(2b) Inspect and adjust the requested scope
(2c) Provide the authorization consent
(3a) Provide authorization code
(3b) Provide authorization code
(4) Request access token
(5) Provide access token
(6) Inspect token

Fig. 6: Integration with OAuth 2.0 protocol flow

We have implemented the second option and deployed it using *user-centric* deployment model at the data security gateway (Section III-A). We have integrated authorization descriptor in the OAuth 2.0 *scope* parameter as a Base64 encoded string, which is conformant with the specification and allows the referencing of multiple scopes.

The last step, token inspection (6), is not included in the core OAuth specification [2], however, it is augmented by the accompanying RFC 7662 [25]. As the proposed scope

[5] Such as browser plugin deployed on the client side

structure is fully compatible with existing specifications, we consider this requirement to be satisfied.

V. Application and Discussion

A. Requirements assesment

In Table III we present the comparison of different frameworks with regards to the requirements in Section II-D. For this purpose we examine the proposed authorization descriptor (AD), considering it as an extension of DASP framework, as it relies on its components. We use ✳ to denote the partial fulfillment of a requirement.

Being oriented at a single enterprise and its processes, XACML [9] does not specify the interactions with clients to request the authorization from the user (R1) and to claim constraints (R3). Instead, its *policy enforcement point* (PEP) dynamically intercepts accesses and generates authorization requests for *policy decision point* (PDP) to calculate authorization decisions for events using a predefined set of security policies. The clients hence do not have structured means to express their authorization requirements or acceptable constraints, nor to obtain them in the online flow. Being inspired by XACML architecture, UMA [3] similarly requires the existence of *a priori* policies to authorize requests. It, however, specifies a *claims-gathering flow*, which allows the elevation of client privileges. The reliance on security policies allows the frameworks to fulfill (R2). Due to the focus on a single organization, the granular specification of accesses does not scale well beyond organizational boundaries.

AD smoothly integrates authorization scope with service model to support (R4) in a way that allows agents to automatically derive structure and extraction rules for previously unknown services. Although they support this requirement, XACML and UMA impose additional implementation overhead. This is due to the need to separately implement resource models for each service, which does not scale well for web-scale integrations.

The online transformation of resources (R5) may be supported in XACML using *obligations*. They, however, depend on particular implementation, do not scale well and cannot be reused across environments. SUNFISH [26] provides a set of predefined transformational functions relevant for the security that can be referenced and reused by different participating organizations. For the same reason, the federation framework allows establishing scalable management (R6), but only for the federation members that fulfill technical and formal requirements established in an out-of-the-band process.

Finally, (R7) assumes the *semantic interoperability* [16] to support scalable integration across environments. Pursuing the interoperability on *syntactic* level only, as envisaged in other frameworks, introduces the obstacles for integration beyond organizational boundaries (Section III-D).

B. Cooperative authorization establishment

Presented real-world example (Section IV-A) demonstrates the capability of the proposed structure to support the expression and refinement of the authorization requirements through the multilateral interaction chain.

We assume that the client may be willing to provide fine-grained, correct view of its access requirements that

TABLE III: Specification and management of authorizations

Req.	XACML [9]	SUNFISH [26, 27]	OAuth 2 [2]	UMA [3]	AD
(R1)	✗	✗	✓	✳	✓
(R2)	✓	✓	✗	✓	✓
(R3)	✗	✗	✗	✗	✓
(R4)	✓	✓	✗	✗	✓
(R5)	✳	✓	✗	✗	✓
(R6)	✗	✳	✗	✗	✓
(R7)	✗	✗	✗	✗	✓

optimally conforms to security and privacy expectations of the resource owner. However, the expected degree of cooperative authorization establishment may vary along the lines of completeness and user's security goals for different reasons. In some cases, clients can demonstrate low willingness to cooperate, they could act from the overhead-reducing perspective, or simply they could behave with the malicious intent. In other cases, due to the strict access control, clients could be prevented from retrieving the complete information on the organization of resources or their values, which may be necessary to structure the optimal authorization scope.

In any case, the user should still be given the opportunity to manage the extent of given authorizations. We call this *balancing between security and utility*, as our proposal aims to enrich existing simplified security controls with adaptive and expressive structures that allow a high level of granularity along with the contextual and dynamic provision of access permissions.

C. Enforcement and the integration with systems

While the definition of given authorizations is one part of the security management process, the other part includes their actual enforcement. Typically, these activities are coupled to a single organization as they depend on its internal models, interfaces and the infrastructure. As a result, many cross-organizational interactions depend on hard-coded configurations and interfaces that need to be adjusted to operate with each organization individually.

Our proposal approaches this problem primarily from the perspective of the specification of the requirements, contributing with the interoperability layer that allows interconnecting the models from different environments. Nevertheless, the organizations have and prefer to implement enforcement infrastructure according to their internal processes and information models. However, by considering an additional interoperability layer, they can benefit by allowing their clients a higher degree of usability and control, which increases the value and the reach of their services. Due to the flexible approach and the possibility to express interfaces, resources, and operations at arbitrary levels of detail, service providers are free to choose the degree of implementation that provides the optimal intersection of user value and the integration overhead.

The integration overhead to create and derive the proposed structure is low, considering its reliance on existing formats. There are different libraries available to support RDF, JSON-LD or Turtle syntaxes. In addition to that, in our other work, we introduce data security gateway [17], which provides the policy-based tool to manage and enforce security controls of Web APIs (Section III-A). Augmented with the flows and the structure presented in the current work, this gateway provides an out-of-the-box solution that can be applied over existing APIs to reduce the implementation and integration overhead.

VI. Related work

Several initiatives have been focused on the development and application of open vocabularies on the web with different goals, including Hydra vocabulary [28] for the creation of generic API clients, integration of IoT devices [29] or overall consolidation of linked vocabularies [30]. Our work is complementary to these approaches as it aims to fill the gap for specifications in the security domain.

Hüffmeyer and Schreier proposed RestACL language for the protection of RESTful services [31]. Alam et al. present xDAuth framework [32] that supports authorization and delegation in the RESTful service architecture, which relies on XACML as policy language [9]. Both of these approaches consider the perspective of a single enterprise, providing the access specification capabilities on a granularity level of a resource. SUNFISH [27] is a recent EU-supported initiative to develop a framework for the establishment of secure cloud-based service federations. SUNFISH externalizes authorization beyond a single organization by providing consolidated policy model and out-of-the-box support for dynamic resource transformation in the federation. It, however, focuses on traditional web services and imposes high adoption barriers for entities beyond public administrations.

VII. Conclusion

In this work, we introduced management flows and authorization descriptor that support cooperative and adaptive authorization in heterogeneous environments. The proposed structure provides a high-degree of expressivity, enabling the specification of rich, granular and contextual security requirements and restrictions that support the dynamic transformation of resources in Web API integrations. Our contribution aims to advance the manageability of security controls in the cloud by providing self-dereferenceable and transparent structures that allow resource- and operation-aware specification of authorizations. By introducing the semantic interoperability layer, we establish the link between service models and security controls, which allows their tighter integration and facilitates reuse and transparency of controls both in a closed system or across different environments.

Acknowledgment

This work has been partially supported by A-SIT Secure Information Technology Center Austria and EU H2020 Programme under the SUNFISH project, grant No. 644666.

References

[1] M. Garriga, C. Mateos, A. Flores, A. Cechich, and A. Zunino, "RESTful service composition at a glance: A survey," *Journal of Network and Computer Applications*, vol. 60, 2016.

[2] D. Hardt, "The OAuth 2.0 authorization framework," 2012.

[3] T. Hardjono *et al.*, "User-managed access (UMA) profile of OAuth 2.0," *Internet Engineering Task Force (IETF)*, 2015.

[4] F. Bülthoff and M. Maleshkova, "RESTful or RESTless–Current state of today's top Web APIs," in *European Semantic Web Conference*. Springer, 2014, pp. 64–74.

[5] B. Suzic, "Security Aspects of Web-APIs," Tech. Rep., 2017.

[6] ——, "Securing integration of cloud services in cross-domain distributed environments," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016.

[7] M. Vukovic *et al.*, "Riding and thriving on the API hype cycle," *Communications of the ACM*, vol. 59, no. 3, pp. 35–37, 2016.

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

[9] O. X. T. Committee *et al.*, "eXtensible Access Control Markup Language (XACML) Version 3.0," *OASIS*, 2013.

[10] A. Mourad and H. Jebbaoui, "SBA-XACML: set-based approach providing efficient policy decision process for accessing web services," *Expert Systems with Applications*, vol. 42, 2015.

[11] M. Marian, "iPaaS: Different ways of thinking," *Procedia Economics and Finance*, vol. 3, pp. 1093–1098, 2012.

[12] K. Guttridge *et al.*, "Magic quadrant for enterprise integration platform as a service," Gartner Inc., 2017.

[13] F. B. Schneider, "Least privilege and more [computer security]," *IEEE Security & Privacy*, vol. 99, no. 5, pp. 55–59, 2003.

[14] S. Farrell, "API Keys to the Kingdom," *IEEE Internet Computing*, vol. 13, no. 5, 2009.

[15] M. Jones and D. Hardt, "The OAuth 2.0 authorization framework: Bearer token usage," Tech. Rep., 2012.

[16] H. van der Veer and A. Wiles, "Achieving technical interoperability," *ETSI*, 2008.

[17] B. Suzic, "User-centered security management of API-based data integration workflows," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016.

[18] R. Verborgh *et al.*, "Bottom-up web apis with self-descriptive responses," in *Proceedings of the First Karlsruhe Service Summit Workshop-Advances in Service Research*, 2015, p. 143.

[19] B. Suzic, "LOD and LOV for Authorization Concepts," Tech. Rep., 2017.

[20] R. J. Brachman *et al.*, "Krypton: A functional approach to knowledge representation," *Computer*, vol. 10, 1983.

[21] R. Studer *et al.*, "Knowledge engineering: principles and methods," *Data & knowledge engineering*, vol. 25, no. 1-2, 1998.

[22] K. Janowicz, F. Van Harmelen, J. A. Hendler, and P. Hitzler, "Why the data train needs semantic rails," *AI Magazine*, 2014.

[23] W. O. W. Group, "OWL 2 Web Ontology Language Document Overview (Second Edition)," W3C, 2012.

[24] D. Roman *et al.*, "Wsmo-lite and hrests: Lightweight semantic annotations for web services and restful apis," *Web Semantics: Science, Services and Agents on the WWW*, vol. 31, 2015.

[25] J. Richer, "OAuth 2.0 Token Introspection," 2015.

[26] F. P. Schiavo, V. Sassone, L. Nicoletti, and A. Margheri, "FaaS: Federation-as-a-Service," *arXiv:1612.03937*, 2016.

[27] B. Suzic *et al.*, "Balancing utility and security: Securing cloud federations of public entities," in *OTM Confederated International Conferences*. Springer, 2016, pp. 943–961.

[28] M. Lanthaler and C. Gütl, "Hydra: A vocabulary for hypermedia-driven web apis." *LDOW*, vol. 996, 2013.

[29] A. Gyrard *et al.*, "Reusing and Unifying Background Knowledge for IoT with LOV4IoT," in *Future Internet of Things and Cloud, IEEE 4th International Conference on*. IEEE, 2016.

[30] P.-Y. Vandenbussche *et al.*, "Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the Web," *Semantic Web*, vol. 8, no. 3, pp. 437–452, 2017.

[31] M. Hüffmeyer and U. Schreier, "RestACL: An Access Control Language for RESTful Services," in *Proceedings of the 2016 ACM International Workshop on ABAC*. ACM, 2016.

[32] M. Alam *et al.*, "xDAuth: a scalable and lightweight framework for cross domain access control and delegation," in *Proceedings of the 16th ACM SACMAT Symposium*. ACM, 2011.