Federated Authorization for Software-as-a-Service Applications

Maarten Decat[1], Bert Lagaisse[1], Dimitri Van Landuyt[1], Bruno Crispo[2], and Wouter Joosen[1]

[1] iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
first.last@cs.kuleuven.be
[2] Department of Information Engineering and Computer Science, University of Trento, Trento, Italy
crispo@disi.unitn.it

Abstract. Software-as-a-Service (SaaS) is a type of cloud computing in which a tenant rents access to a shared, typically web-based application hosted by a provider. Access control for SaaS should enable the tenant to control access to data that are located at the provider based on tenant-specific access control policies. To achieve this, state-of-practice SaaS applications provide application-specific access control configuration interfaces and as a result, the tenant policies are evaluated at the provider side. This approach does not support collaboration between provider-side and tenant-side access control infrastructures, thus scattering tenant access control management and forcing the tenant to disclose sensitive access control data. To address these issues, we describe the concept of federated authorization in which management and evaluation of the tenant policies is externalized from the SaaS application to the tenant. This centralizes tenant access control management and lowers the required trust in the provider. This paper presents a generic middleware architecture for federated authorization, describing required extensions to current policy languages and a distributed execution environment. Our evaluation explores the trade-off between performance and security and shows that federated authorization is a feasible and promising approach.

Keywords: Federation, authorization, access control, Software-as-a-Service.

1 Introduction

Software-as-a-Service or SaaS is a form of cloud computing in which the *tenant* rents access to a shared application hosted by the *provider* [21]. The tenant is an organization representing multiple end-users, who use the application through a thin client, typically a web browser. A SaaS application is used by multiple tenant organizations and each organization can be tenant for multiple SaaS applications. While traditional SaaS applications such as Google Apps (an office suite) and Salesforce (CRM) mainly target small enterprises, large enterprises

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 342–359, 2013.

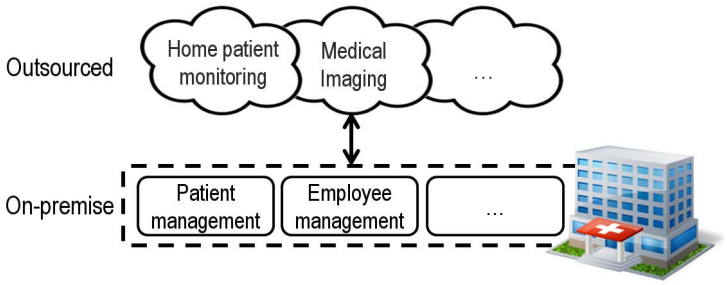


Fig. 1. Large organizations such as hospitals have started to adopt SaaS applications. These organizations often employ on-premise infrastructures for managing their users efficiently. Access control for SaaS should integrate with these on-premise infrastructures, which still poses challenges.

have recently started to adopt SaaS as well, for example in the domains of document processing [6], workforce management [6] or e-health [4,5]. This evolution stresses key challenges, such as the increased importance of security in general and access control in particular.

While the data in a SaaS application is hosted by the provider, the tenant should still be able to control access to it based on tenant-specific access control policies. While large organizations often employ on-premise access control infrastructures for managing their users centrally and efficiently (illustrated in Fig. 1), state-of-practice SaaS applications offer tenants an application-specific access control configuration interface. As a result, the tenant policies are stored and evaluated provider-side. This causes two major problems: (i) This approach fails to integrate with the on-premise infrastructures of the tenant and again distributes and scatters its access control management. Since a single organization can be tenant of multiple SaaS applications, this leads to large administrative overhead and eventually to inconsistencies and security holes. (ii) This approach forces the tenant to disclose to the provider all access control data required for evaluating its policies. While SaaS applications outsource specific functionality, the organization-wide policies of the tenant apply. These policies reason about data of the organization stored in the on-premise infrastructures, such as attributes of a patient or a physician. Although the tenant may trust the provider with the data in the application, it does not necessarily want to trust the provider with this sensitive on-premise data needed for access control. Moreover, regulatory requirements such as the European DPD [11] even forbid the tenant to share the data.

To address the problems identified above, we describe the notion of *federated authorization*. In analogy to federated authentication [1,2], federated authorization externalizes tenant policy evaluation from the provider to the tenant. This approach fully centralizes the access control management of the tenant and enables the tenant to enforce policies on the SaaS application without disclosing the sensitive data that are required to evaluate the policies.

This paper first introduces the concept of federated authorization for SaaS applications. The paper then presents a generic middleware architecture for federated authorization in which the XACML policy language is extended and a supporting distributed execution environment is defined. This architecture has three key features: (i) requesting an access control decision from the tenant, (ii) handling local and remote attributes and (iii) handling local and remote obligations. The paper then evaluates federated authorization in terms of performance based on a prototype of the middleware and finally discusses the trade-off between performance and security, showing that the notion of federated authorization is a feasible and promising approach. While similar approaches to federated authorization exist in related work (e.g., [7,8,19]) and its essence is used in practice in specific domains (e.g., 3-D Secure for internet payments [3]), this work focuses on a generic, policy-based middleware architecture. While this work originated from the domain of SaaS, we believe that federated authorization is an important enabler for all cross-organizational applications.

The remainder of this paper is structured as follows. Section 2 describes the context of this work, i.e., access control and federation techniques. Section 3 elaborates on the problem statement based on a realistic case study in the domain of e-health. Section 4 describes the concept of federated authorization and presents our supporting architecture and language extensions. Section 5 evaluates federated authorization in terms of performance based on our prototype and Section 6 discusses the results and the impact on security. Section 7 gives an overview of the related work and Section 8 concludes this paper.

2 Background

This work fits in the domain of federated access control, applied to SaaS applications. Since SaaS was introduced in the previous section, this section discusses access control and support for federation.

2.1 Access Control

Access control is an important part of application-level security that limits the *actions* which a *subject* (e.g., an end-user) can take on an *object* in the system (e.g., a file). The process of access control is generally divided into authentication and authorization: authentication confirms the stated identity of a subject and authorization confirms that the authenticated subject is allowed to do the desired action on the object by evaluating *access control policies*. Several models have been proposed for expressing these, of which attribute-based access control (ABAC, [16]) employs key-value properties called *attributes* of the subject (e.g., the subject id, a username or a role), the object (e.g., the object id, location or content) and the environment (e.g., the time, physical location or usage context).

The reference architecture for policy-based authorization middleware was defined by IETF and DMTF and refined by the XACML standard [22]. In the reference architecture (see Fig. 2), the policy decision point (PDP) makes the

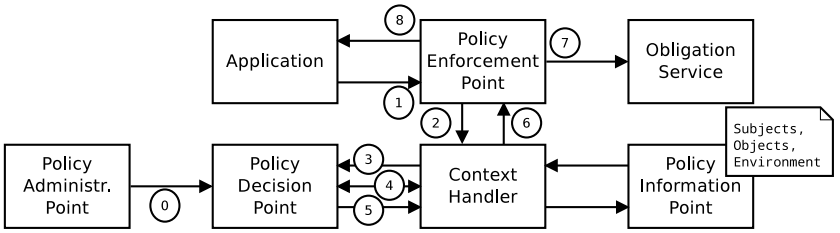


Fig. 2. The XACML reference architecture for policy-based authorization middleware [22]

actual authorization decision. The policy enforcement point (PEP) is integrated in the application (e.g., as an API or a reference monitor) and requests an access control decision from the PDP through the context handler. A decision request generally consists of information about the subject, the object, the action and the environment. The context handler gathers required attributes from one or more policy information points (PIPs, e.g., a database), which the PDP uses to evaluate the applicable policies loaded from the policy administration point (PAP). If needed, the PDP can request additional attributes from the context handler. Finally, the PDP returns its response to the PEP. Such a response consists of the access control decision itself (permit or deny) and possibly obligations. An obligation represents an action to be executed after the access control decision, e.g., updating an attribute such as the subject's usage quota. The PEP fulfills the obligations using the obligation service and enforces the PDP's decision.

2.2 Support for Federation

The XACML reference architecture is aimed at access control systems within a single organization. For applications in which multiple organizations are involved, *federation techniques* have been developed. A *federation* can be defined as an organizational structure in which multiple organizations have set up collaboration agreements. Each organization represents a separate domain in terms of security and administration and is bound to other domains by the means of (limited) trust. SaaS applications are an example of this. While a federation can in general comprise more than two organizations, this paper focuses on the provider and the tenant.

One federation technique is *federated authentication* [1,2]. For SaaS, federated authentication allows to externalize authentication from the provider to the tenant: the tenant authenticates the subject locally and afterward states his or her identity and attributes to the provider. The advantages are threefold: (i) It gives the tenant full control over the means of authentication. (ii) It gives the tenant full control over the access control data shared with the provider. (iii) It effectively centralizes user management at the tenant side, offering scalable user administration to the tenant.

Supported by federated authentication, two main strategies for authorization exist in state-of-the-art access control for SaaS: (i) Full provider-side authorization:

In this case, all access control data and policies are located at the provider side. The provider evaluates the policies based on local data and no federation techniques are used. (ii) Provider-side authorization with federated authentication: In this case, the access control data is centralized at the tenant side, but the policies are located at and evaluated by the provider and the tenant data is shared using federation techniques. In both strategies, only the provider evaluates access control policies.

3 Problem Elaboration and Illustration

This section presents the problem statement of this paper. First, Section 3.1 presents an illustrative case study in the domain of e-health. From this, we derive three key requirements and argue their relevance for SaaS in Section 3.2.

3.1 Case Study in e-Health

As stated in the introduction, large enterprises have started to adopt SaaS. An example of such an application inspired by a number of research projects [4,5] is a home patient monitoring system provided to hospitals as a service (see Fig. 3). The system allows patients to be monitored continuously after leaving the hospital, for example by a chest band or a wrist sensor. The measurements (the application data) are sent from the patients to the application back-end using a smart-phone as an intermediary device. The measurements are stored and processed by the provider: telemedicine operators continuously check the patient's status and the patient's physician at the hospital is notified in case of important evolutions. A patient's status can also be viewed by other physicians and nurses and by the patients themselves. The hospitals are the tenants of the application and each tenant represents multiple patients and physicians, i.e., the end-users of the application. Next to the monitoring system, the hospital employs other on-premise applications, e.g., for patient management, and other SaaS applications, e.g., for medical imaging.

Example policy. A typical example of an organization-wide policy employed by the hospital in this case study can be summarized as follows: "*a physician*

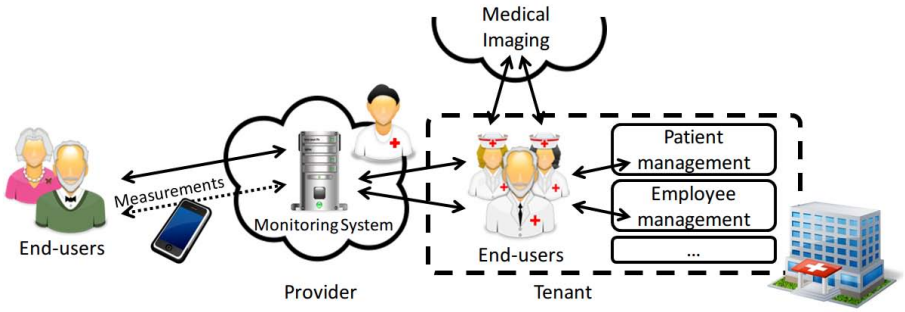


Fig. 3. The case study that inspired this work: a home patient monitoring system offered to hospitals as a SaaS application

can only view or alter patient data when currently treating the patient who owns the data and if that patient has given consent or when the data is relevant to his specialization or when the patient is in a life-threatening situation".

3.2 Access Control Requirements

It is clear that the provider wants to control access to the application. The provider constrains its own end-users, i.e., the telemedicine operators, and constrains its tenants, e.g., to ensure a tenant has enough credit to perform a certain action or is credited afterward. However, e-health applications are subject to regulatory requirements (e.g., the European DPD [11]) and even if the data is hosted by the provider, the hospital is still accountable for it. Therefore, the hospital has to be able to control access to the application as well and the hospital-wide policy presented above also applies to the SaaS application. In this work we take the point of view of the tenant and focus on the following three requirements:

Scalable administration. Hospitals can have a medical staff of thousands and treat even more patients each day, continuously producing large amounts of new application data. Manual security administration on this scale would incur too much administrative overhead and would quickly lead to inconsistencies and security holes. To avoid this, hospitals have extensive centralized security infrastructures at hand, e.g., organization-wide patient management systems. When employing SaaS applications such as the patient monitoring system, this degree of centralization should be upheld and security administration should remain centralized at the hospital.

Complex and large policies. Current e-health policies require detailed and extensive information. For example, the small policy presented above requires user roles, treating relationships, patient consent, resource content and more. Furthermore, the complexity of current policies makes it impossible to statically determine the required data up-front.

Sensitive access control data. Some of the access control data required for evaluating the policy presented above is sensitive in nature, such as the list of patients being treated by a physician, patient diseases or patient consent. While the hospital may trust the provider with the monitoring data in the application, it does not necessarily want to trust the provider with this sensitive access control data. Moreover, regulatory requirements such as the European DPD [11] even forbid the hospital to share this data.

As mentioned in Section 2.2, the hospital policies are evaluated by the provider in state-of-the-art access control for SaaS. As a consequence, the hospital policies are still distributed and fragmented over the multitude of applications it uses. Moreover, all tenant data required for evaluating the tenant policies has to be shared with the provider. Therefore, state-of-the-art federated access control does not fulfill the requirements stated above: it leads to limited administrative scalability and forces the tenant to disclose sensitive access control data.

4 Solution: Federated Authorization

We address the problems identified in the previous section by externalizing tenant policy evaluation from the provider to the tenant. This effectively centralizes tenant access control management and enables the tenant to enforce policies on the SaaS application without having to share the access control data needed for evaluating these. In analogy to federated authentication, we call this *federated authorization*.

Section 4.1 first describes the key features required for realizing federated authorization and their impact on the XACML reference architecture and current policy languages. Section 4.2 presents our generic architecture for federated authorization and finally Section 4.3 illustrates the required policy language extensions using the XACML policy language.

4.1 Key Features

With federated authorization, the provider evaluates its own policies while the evaluation of tenant policies is externalized from the provider to the tenant. With respect to the XACML reference architecture (see Section 2.1), federated authorization adds three key features: (i) requesting an access control decision from the tenant, (ii) handling local and remote attributes and (iii) handling local and remote obligations. For each of these new features, we determine what should be added to the XACML reference architecture and to current policy languages such as XACML [22], Ponder [12] or EPAL [10].

Requesting Tenant Access Control Decisions. As a first key feature, the provider is able to request an access control decision from the tenant concerning the SaaS application. Afterward, the provider combines the tenant response with its own policy results so that the requested action is only permitted if both parties permit it. Allowing the provider to request a tenant access control decision impacts both the architecture and the policy language.

Architecture. The tenant should provide a service to receive decision requests from the providers of the SaaS applications it employs. Such a request should identify the provider and should contain information about the subject, the object, the action and the environment, similar to a request from PEP to PDP. Using this information, the tenant determines and evaluates the applicable policies and returns its response. This response contains the decision itself (permit or deny) and possibly obligations, similar to a response from PDP to the PEP. Notice that the provider does not reference a particular tenant policy, but rather the tenant as a whole behaving like a PDP.

Policy Language. The policy language should allow the provider to refer to the access control decision service of the tenant and specify how the result should be processed, similar to on-premise policies.

Handling Local and Remote Attributes. As a second key feature, all required attributes are made available to the respective policies. As mentioned

before, the provider reasons about its end-users (e.g., the telemedicine operators in the case study) and its tenants (e.g., the hospital) and the tenant reasons about its end-users (e.g., the hospital physicians and nurses). For the provider policies, the subjects are therefore its end-users and tenants, the objects are the application data and the environment is the SaaS application. Thus, all attributes required by the provider policies are available locally. However, for a tenant policy, the subjects are its end-users, the objects are the application data and the environment comprises both the SaaS application and the local infrastructure. Data about the tenant end-users and local infrastructure is stored in its on-premise applications, while the rest is hosted by the provider. Thus, the attributes required by the tenant policies are distributed over tenant and provider. Allowing the tenant to handle both local and remote attributes impacts both the architecture and the policy language.

Architecture. For evaluating the tenant policies tenant-side, the attributes of the objects in the SaaS application and the attributes of the provider-side environment have to be made available to the tenant. In addition, while attributes can be added to the initial request to the tenant, the complexity of current policies makes it impossible to statically determine the required attributes up-front. Therefore, the provider should provide a service to the tenant to dynamically fetch required attributes.

Policy Language. When evaluating the tenant policies, the context handler should know where to find the required attributes. Therefore, the policy language should allow to define the location of each attribute referenced in the policies.

Handling Local and Remote Obligations. As a third key feature, the tenant is able to handle both tenant-side obligations, e.g., for logging, and provider-side obligations, e.g., for updating the access control history of an object. Allowing the tenant to handle local and remote obligations impacts both the architecture and the policy language.

Architecture. The response from the tenant policy evaluation service should contain the obligations specified by the tenant which will be fulfilled by the provider. This was already mentioned before. The tenant response should not contain locally fulfilled obligations.

Policy Language. The policy language should allow the tenant to specify where obligations should be fulfilled: locally or remotely.

4.2 Generic Middleware Architecture for Federated Authorization

Based on the architectural requirements listed above, we now present a generic architecture for federated authorization. We first describe the decomposition of the architecture aligned to the XACML reference architecture presented in Section 2.1 and then illustrate the resulting access control flow.

Architecture Decomposition. Figure 4 shows the decomposition of the architecture. First of all, the provider hosts the SaaS application and therefore also the

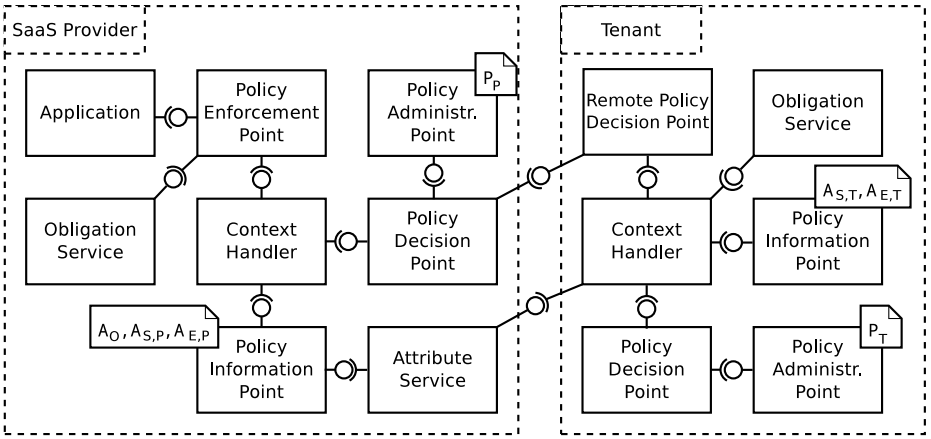


Fig. 4. The generic architecture for federated authorization. P_P and P_T are the provider and tenant policy sets respectively and A_O, $A_{S,P}$, $A_{E,P}$, $A_{S,T}$ and $A_{E,T}$ are as defined in Section 4.2.

PEP, no application components are located at the tenant side. Since both parties evaluate policies and process obligations, both have a PAP, a PDP, a context handler, one or more PIPs and an obligation service. The provider PIP contains the attributes of the objects in the SaaS application (A_O), the attributes of the provider policy subjects ($A_{S,P}$) and the attributes of the provider-side environment ($A_{E,P}$). The tenant PIP contains the attributes of the tenant policy subjects ($A_{S,T}$) and the attributes of the tenant-side environment ($A_{E,T}$). For handling decision requests, the tenant offers a Remote Policy Decision Point (RPDP) to the provider. For handling attribute requests, the provider offers an attribute service to the tenant. The provider PDP is extended with functionality to contact the RPDP and the tenant context handler is extended with functionality to contact the provider attribute service. To summarize, the resulting interface between provider and tenant consists of two services: the tenant policy evaluation service and the provider attribute service. Notice that the architecture still allows the tenant to use its infrastructure for on-premise applications as well, which is not shown in the figure.

Access Control Flow. Figure 5 illustrates the resulting access control flow. The presented flow starts after the provider and tenant policies are loaded from their respective PAPs and the end-user has been successfully authenticated. The remainder of the flow is as follows: (1) With each request an end-user makes to the application, the PEP constructs an access control request and forwards this to the local context handler. (2) The context handler collects statically known attributes from the local PIPs (only one is shown in Fig. 5), adds these to the request and sends it to the local PDP. (3) The PDP determines the applicable provider policies and evaluates them, thereby dynamically requesting attributes from the context handler. When the PDP encounters a remote policy reference,

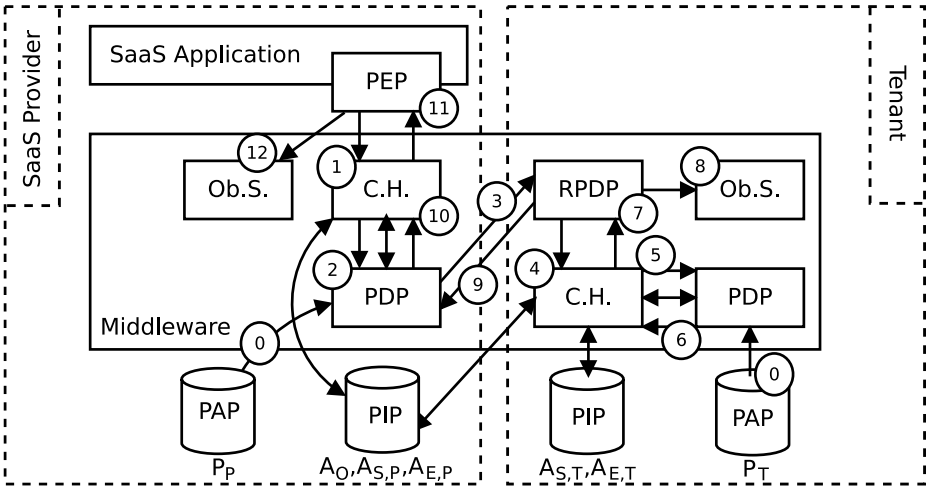


Fig. 5. The resulting access control flow for the generic architecture. C.H. is context handler, Ob.S. is obligation service, P_P and P_T are the provider and tenant policy set respectively, PEP, PDP, PIP and PAP are as defined in Section 2.1 and A_O, $A_{S,P}$, $A_{S,T}$, $A_{E,P}$ and $A_{E,T}$ are as defined in Section 4.2. For readability reasons, the provider attribute service is not shown explicitly.

it asks the context handler to determine how and where to contact the tenant. The PDP then constructs a decision request and sends it to the tenant RPDP. (4) From the tenant point of view, the RPDP acts similarly to a PEP and forwards the request to the tenant context handler. (5) The context handler collects statically known attributes from the local PIPs (only one is shown in Fig. 5), adds these to the request and forwards it to the PDP. (6) The PDP determines the applicable tenant policies and evaluates them, thereby dynamically requesting attributes from the context handler. The context handler fetches tenant attributes locally and contacts the provider attribute service for provider attributes. The tenant PDP eventually returns its response (i.e., decision and obligations) to the context handler. (7) The context handler returns the response to the RPDP. (8) The RPDP fulfills tenant obligations using the tenant obligation service and removes these from the response. (9) The RPDP returns the resulting response to the provider PDP. (10) The provider PDP combines the tenant decision with the result of the provider policies so that the request is only allowed if both parties allow it and returns the overall response to the provider context handler. (11) The provider context handler returns the response to the PEP. (12) The PEP fulfills the obligations using the provider obligation service and enforces the decision.

4.3 Extensions to Current Policy Languages

The three key enablers to support federated authorization identified in Section 4.1 all required policy language extensions. To illustrate the practical impact of these requirements, we have extended the XACML policy language v2 [22]. We opted for XACML because of its wide-spread use in both academia and industry and its active development. In this section we describe the extensions we introduced, their specifications are provided on-line[2]. We start by introducing XACML first.

The XACML policy language. In addition to the access control reference architecture, the XACML standard also defines a policy language [22], hereafter simply referred to as "XACML". XACML expresses policies using XML. Three main elements are defined: `<PolicySet>`, `<Policy>` and `<Rule>`. A policy set can contain multiple policies and a policy can contain multiple rules. A rule specifies an effect (permit or deny), attribute-based conditions for this to hold and obligations to fulfill with it. A policy combines the results of its rules using a rule-combining algorithm (e.g., deny overrides), a policy set combines the results of its policies using a similar policy-combining algorithm. Each of these elements also contains an attribute-based target specifying in which situations it applies.

Referencing remote policies. For referencing remote policies, the `<RemotePolicyReference>` element is introduced. As mentioned before, the tenant behaves as a remote policy from the provider point of view. Therefor, evaluating a remote policy reference returns a policy evaluation result and the element can be part of a policy set, similarly to the `<Policy>` element. The `PolicyId` attribute specifies the id of the remote policy. Following the XACML design principles, it is left to the context handler to determine how and where to contact it. A remote policy reference can also contain a description, a target and obligations for local use.

Handling local and remote attributes. For differentiating between local and remote attributes, we follow the XACML design choice of having the context handler infer the location of an attribute based on its id instead of defining attribute properties declaratively. Thus, XACML is not extended for this requirement.

Handling local and remote obligations. For differentiating between tenant and provider as obligation targets, the `<Obligation>` element is extended with the optional `FulfillWhere` attribute. This attribute specifies whether the obligation should be fulfilled locally (with the tenant) or remotely (with the provider), local fulfillment being the default. The new attribute is only to be used in tenant policies. It is the responsibility of the tenant to remove local obligations from its response so that the provider can interpret all obligations as before.

5 Performance Evaluation

Following the description of the supporting middleware, this section evaluates the concept of federated authorization in terms performance based on a prototype of the middleware. The performance evaluation explores the impact of federation on the time it takes for the provider to reach an access control decision. Because

federated authorization only affects the evaluation of the tenant policies, the provider policies are not taken into account.

Test Setup. The tests compare federated authorization against the two strategies for SaaS authorization in state-of-the-art mentioned in Section 2.2. Thus, three different cases of authorization are compared: full provider-side authorization, provider-side authorization with federated authentication and federated authorization. Notice that only federated authorization keeps sensitive access control data of the tenant confidential.

The tests employ policies which require 10, 20 and 30 attributes. Access control studies in a number of research projects [4,5,6] show that these numbers represent modest to large policies. Because multiple strategies for fetching attributes exist, ranging from one-at-a-time to combined requests, the tests also compare the two extreme strategies: fetching all required attributes separately and fetching all attributes at the same time as one multi-valued attribute. All attributes are fetched just-in-time and no attribute caching is used.

Prototype. The prototype of the middleware instantiates the architecture described in Section 4.2. The prototype extends the SunXACML policy evaluation engine[1] with the new `<RemotePolicyReference>` element. Attributes are stored in SQLite databases[1]. Cross-organization communication is realized out-of-band using SAML [1] and the SAML profile of XACML [20] over SOAP web-services[1] implemented on top of Apache Tomcat 7[1] using the Apache CXF services framework[1] and the OpenSAML Java library[1]. For similarity, both the provider and the tenant PDP are run on top of Tomcat. In order to focus on policy evaluation time, the prototype also omits channel encryption or authentication. The prototype (6KLOC) is publicly available[2].

The prototype contains four main components: (i) the provider PDP, (ii) the tenant PDP (iii) the provider attribute web service and (iv) the tenant attribute web service. Each of these components is run on a separate machine with 4GiB RAM and two cores of 2.40GHz running Ubuntu 12.04. Local databases are run on the same node as the services that use them. To simulate the distance between tenant and provider in a realistic SaaS setting, a fixed single-way network delay of 5ms between tenant and provider is applied. Tests are run sequentially and PDP evaluation is done single-threaded. Each test starts with five warm-up requests and is repeated until the confidence interval lies within 1% of the sampled mean for a confidence level of 95%.

Results. The whole set of results is publicly available[2]. Fig. 6 shows the decision time in terms of the number of single-valued attributes. If the policy only requires tenant attributes (Fig. 6a), provider-side authorization performs significantly worse since each attribute is fetched separately the moment the PDP requires it

[1] `http://sourceforge.net/projects/sunxacml/`, `http://www.sqlite.org/`, `http://cxf.apache.org/`, `http://tomcat.apache.org/`, `https://wiki.shibboleth.net/confluence/display/OpenSAML/`

[2] `http://people.cs.kuleuven.be/~maarten.decat/doa-trusted-cloud-2013/`

and each query takes about 20ms as a result of the network latency and XML operations. If the policy only requires provider attributes (Fig. 6b), federated authorization performs worse; full provider-side authorization and provider-side authorization give the same results since all attributes are hosted by the provider.

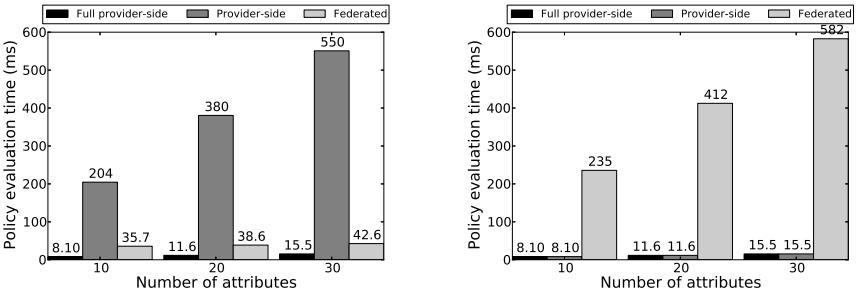
Fig. 7 combines many tenant and provider attributes and shows the decision time in terms of the percentage of tenant attributes in a total of 30 attributes. The figure shows that full provider-side authorization performs best in all cases since no remote queries are required and that federated authorization outperforms provider-side authorization when a little more than half of the attributes are tenant attributes. The asymmetry is a consequence of the extra request needed with federated authorization with respect to provider-side authorization.

Fig. 8 shows the decision time in terms of attribute size, i.e., the number of values in a single attribute. In this case, we employ policies which require a single attribute with 10, 20 or 30 values. If the policy only requires a tenant attribute (Fig. 8a), full provider-side authorization again performs best and both other cases perform similarly since one query between provider and tenant is required. The difference between provider-side and federated authorization is due to the complexity of the tenant-side operation: attribute fetch vs policy evaluation. If the policy only requires a provider attribute (Fig. 8b), federated authorization performs significantly worse than the others, since only this case requires remote queries. In all cases, the decision time remains constant with the size of the required attribute. The absolute difference between Fig. 8a and Fig. 8b is due to two requests instead of one. The absolute difference between many and large attributes in the full provider-side case is due to the fact that the number of required XML translations grows linearly with the number of attributes. Since these numbers are constant with respect to the attribute size, we did not combine them as in Fig. 7.

From these results, we can conclude that federated authorization comes with a performance penalty compared to full provider-side authorization. However, depending on the relative amount of tenant attributes in the tenant policies, federated authorization can achieve better performance than provider-side authorization with federated authentication. To illustrate a realistic case, the example policy rules from the e-health case study presented in Section 3.1 require significantly more tenant attributes than provider attributes: the tenant hosts the subject roles, treating relationships, patient consent and patient diseases while the provider hosts ownership relations and the application data itself.

6 Discussion

In this paper, we described federated authorization. The two main goals of federated authorization were (i) to enable scalable access control management for SaaS applications by integrating it with the on-premise tenant infrastructures and (ii) to lower the required trust in the provider by removing the need to share sensitive access control data. Federated authorization effectively achieves both by allowing the tenant to evaluate its policies locally, but does come with a performance penalty, as shown in Section 5. Full provider-side authorization



(a) Many tenant attributes **(b)** Many provider attributes

Fig. 6. Decision time in terms of the number of tenant and provider attributes

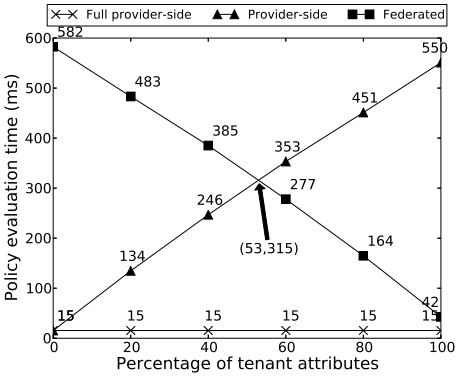
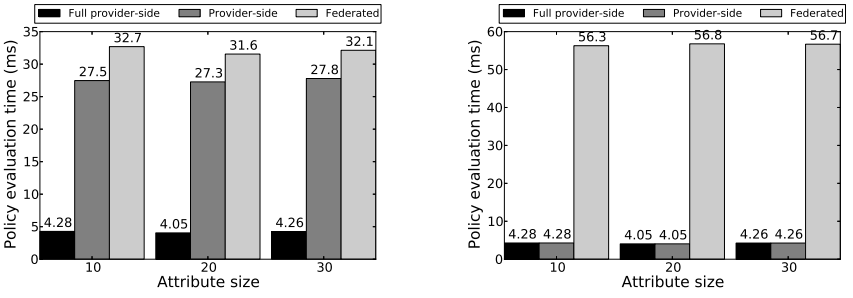


Fig. 7. Policy evaluation time in terms of the percentage of tenant attributes in a total of 30 attributes. The policies from the case study require significantly more tenant attributes than provider attributes, resulting into better performance for federated authorization than provider-side authorization.



(a) Large tenant attribute **(b)** Large provider attribute

Fig. 8. Decision time in terms of the size of provider and tenant attribute

achieves better performance, but does force the tenant to disclose sensitive access control data to the provider. Therefore, we can conclude that federated authorization poses a clear trade-off between security and performance.

While federated authorization removes the need to share sensitive access control data, a potential threat lies in the fact that the provider could still infer information about this data using the collection of access control requests and responses from the tenant collected over time. We argue (i) that the provider has no business incentive for this, since very little can be gained w.r.t. the contract with the tenant and (ii) that the possibly inferred knowledge is limited since both the tenant policies and the access control data used for evaluating these are kept unknown. However, future work is required to answer this question more quantitatively, for example using techniques such as logical abduction. Also notice that externalizing policy evaluation to the tenant allows it to lie or provide incorrect results. However, the tenant has no incentive to do this, since the policies control access to its own data.

Also, while lowering the required trust in the provider, federated authorization does introduce new security threats. From the point of view of a network attacker, the main difference between provider-side and federated authorization is the externalization of the policy evaluation process. This results in a new communication channel between tenant and provider, which introduces a denial of service threat: since a tenant decision is required for every request to the SaaS application, blocking the channel or the services is a way to deny the use of the application for a specific tenant. This threat cannot be easily mitigated. The channel should also be secured against the threats of information disclosure, tampering and spoofing and against attacks such as replay. For SOAP web services, WS-Security [18] provides the necessary security primitives.

Towards the future, the performance of federated authorization can be improved. First of all, the performance evaluation shows that the overhead of federated authorization is highly dependent on the number of requests between provider and tenant. Therefore, a good strategy for fetching attributes is essential to improve performance. For example, combining multiple attributes in a single request is able to lower the number of required requests. In the extreme case, this approaches the tests with a single multi-valued attribute. Secondly, employing caching also has the ability to lower the required provider-tenant communication. However, caching also has an impact on security, as discussed by Gheorghe et al. [15]. Next to attributes, access control decisions can also be cached or even inferred, as shown by Wei et al. [25]. We plan to investigate both in future work. Finally, a more fine-grained and dynamic policy deployment strategy also has the ability to improve performance while maintaining the trust advantages. This paper explored two extremes: the tenant policies are either completely evaluated by the tenant or by the provider. Based on the location of the required data and its sensitivity, the tenant policies could be split and distributed for optimal performance while maintaining the confidentiality of the tenant data. We recently explored this strategy [13] and plan to refine this work in the near future.

7 Related Work

Related to this work, xDAuth, the work of Lischka et al. and the work of Ardagna et al. take on similar requirements. xDAuth [7] provides confidential access control on remote applications by introducing an authorization broker. However, this approach effectively shifts the required trust to a central third party. Lischka et al. [19] introduce the concept of *deductive policies*, which resemble our policy references. While their work focuses on the policy language, this work focuses on supporting middleware and the trade-off between performance and security. Finally, Ardagna et al. [8] build on XACML and SAML to achieve privacy-preserving access control based on anonymous credentials, which can be used for claiming to adhere to a certain policy without disclosing the user's attributes. With respect to their work, this paper investigates this aspect more thoroughly in the context of SaaS.

Apart from federated authorization, a large body of work also exists about federated access control in general, for example in grid computing. Grids comprise federations of resource providers and consumers joined together in *virtual organizations* and access control in this domain also focuses on scalable management. The most relevant for this work is CAS [23]. Similar to federated authorization, CAS allows resource owners to grant access to a community as a whole and lets the community itself manage fine-grained access control, thereby separating both roles in access control management. However, CAS does not allow community policies to reason about the objects located at the provider side, thereby not achieving full federated authorization.

Some alternative approaches to federated authorization exist as well. Firstly, automated policy deployment is an alternative for achieving administrative scalability. For example, Stihler et al. [24] argue that policy writing should be split between provider and consumer and propose a system in which the consumer automatically deploys its policies to the provider. Efforts such as SPML [14] try to standardize an access control configuration interface for SaaS applications, focusing on user management. However, while these techniques provide administrative scalability, all policies are still evaluated by the provider and they do not address the necessary disclosure of sensitive tenant access control data.

Another alternative for achieving confidentiality is encrypting the access control policies and data when sharing them with the provider. This has been explored by Asghar et al. [9] and could theoretically allow the provider to evaluate encrypted policies based on locally available and asynchronously deployed encrypted tenant attributes. However, a large performance overhead is introduced for policy evaluation and the expressiveness of the supported policies is still limited, making federated authorization a more generally applicable technique.

Finally, XML gateways such as IBM Tivoli Access Manager [17] are a third alternative for federated access control on remote services. An XML gateway is placed at the perimeter of the organization and reviews every request to a remote service, similar to a firewall. This approach is also able to enforce tenant access control policies based on local access control data. However, the policies are limited to reasoning about service messages and are mainly used for

transparently adding credentials or for encryption. Moreover, SaaS applications are a good match with mobile users, while XML gateways require every request to originate from the physical network of the tenant.

8 Conclusions

In this paper, we have described the concept of federated authorization, supported by a generic middleware architecture and policy language extensions. Based on a prototype of the middleware, we have shown that the notion of federated authorization is a feasible and promising approach that provides a clear trade-off between security and performance.

This work was motivated in the context of SaaS. While traditional access control focuses on the provider controlling access to its own data, access control for SaaS should also involve the tenant. SaaS requires federated authorization because of the inherently limited trust between tenant and provider: the tenant may trust the provider with the data in the application, but not necessarily with sensitive data required for authorization. Moreover, because an organization can be tenant for a large number of SaaS applications, centralization of tenant access control management is essential to the adoption of SaaS. However, federated authorization is applicable in many distributed applications that involve multiple organizations. In today's growing ecosystem of service-oriented business coalitions, the need for federated access control and for federated authorization in particular will only grow.

Acknowledgements. We would like to thank Wouter De Borger for his in-depth discussions and feedback. This research is partially funded by the Research Fund KU Leuven, by the EU FP7 project NESSoS and by the Agency for Innovation by Science and Technology in Flanders (IWT). With the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE).

References

1. Security Assertion Markup Language (SAML) v2.0 (March 2005), http://www.oasis-open.org/standards#samlv2.0
2. OpenID Authentication 2.0 - Final (December 2007), http://openid.net/specs/openid-authentication-2_0.html
3. 3-D Secure - Wikipedia, the free encyclopedia (July 2013), http://en.wikipedia.org/wiki/3-D_Secure
4. E-Health Information Platforms (E-HIP) (July 2013), http://distrinet.cs.kuleuven.be/research/projects/showProject.do?projectID=E-HIP
5. Healthcare professional's collaboration Space (Share4Health) (July 2013), http://distrinet.cs.kuleuven.be/research/projects/showProject.do?projectID=Share4Health
6. Permission, User Management and Availability for multi-tenant SaaS applications (PUMA) (July 2013), http://distrinet.cs.kuleuven.be/research/projects/showProject.do?projectID=PUMA

7. Alam, M., Zhang, X., Khan, K., Ali, G.: xDAuth: a scalable and lightweight framework for cross domain access control and delegation. In: ACM SACMAT, pp. 31–40 (2011)
8. Ardagna, C.A., De Capitani di Vimercati, S., Neven, G., Paraboschi, S., Preiss, F.-S., Samarati, P., Verdicchio, M.: Enabling privacy-preserving credential-based access control with xacml and saml. In: IEEE CIT, pp. 1090–1095 (2010)
9. Asghar, M.R., Ion, M., Russello, G., Crispo, B.: ESPOON: Enforcing encrypted security policies in outsourced environments. In: 2011 Sixth International Conference on Availability, Reliability and Security, ARES, pp. 99–108. IEEE (2011)
10. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language (EPAL). Technical report, IBM (2003)
11. European Commision. Directive 95/46/EC. Directive of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data (1995)
12. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy specification language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
13. Decat, M., Lagaisse, B., Joosen, W.: Toward efficient and confidentiality-aware federation of access control policies. In: Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing. ACM (2012)
14. Cole, G., et al.: Service Provisioning Markup Language (SPML) Version 2.0. OASIS Committee Specification (2006)
15. Gheorghe, G., Crispo, B., Carbone, R., Desmet, L., Joosen, W.: Deploy, adjust and readjust: Supporting dynamic reconfiguration of policy enforcement. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 350–369. Springer, Heidelberg (2011)
16. Jin, X., Krishnan, R., Sandhu, R.: A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 41–55. Springer, Heidelberg (2012)
17. Karjoth, G.: Access control with IBM Tivoli access manager. ACM TISSEC 6(2), 232–257 (2003)
18. Lawrence, K., Kaler, C., Nadalin, A., Monzillo, R., Hallam-Baker, P.: Web Services Security: SOAP Message Security 1.1 (WS-Security) (2006)
19. Lischka, M., Endo, Y., Sánchez Cuenca, M.: Deductive policies with XACML. In: Proceedings of the 2009 ACM Workshop on Secure Web Services, SWS 2009, pp. 37–44. ACM, New York (2009)
20. Lockhart, H., Parducci, B., Rissanen, E.: SAML 2.0 Profile of XACML, Version 2.0
21. Mell, P., Grance, T.: The NIST definition of cloud computing. National Institute of Standards and Technology 53(6), 50 (2009)
22. Moses, T., et al.: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, 200502 (2005)
23. Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S.: A community authorization service for group collaboration. In: Proceedings of the Third International Symposium on Policies for Distributed Systems and Networks, pp. 50–59. IEEE (2002)
24. Stihler, M., Santin, A.O., Calsavara, A., Marcon, A.L.: Distributed usage control architecture for business coalitions. In: IEEE International Conference on Communications, ICC 2009, pp. 1–6. IEEE (2009)
25. Wei, Q.: Towards improving the availability and performance of enterprise authorization systems. PhD thesis, University of British Columbia (2009)