Balancing Utility and Security: Securing Cloud Federations of Public Entities

Bojan Suzic, Bernd Prünster, Dominik Ziegler,
Alexander Marsalek, and Andreas Reiter

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
{bojan.suzic, bernd.pruenster, dominik.ziegler,
alexander.marsalek, andreas.reiter}@iaik.tugraz.at

Abstract. Following their practical needs and legal constraints, recent application of the cloud paradigm among public administrations has been focused on the deployment of private clouds. Due to the increasing amount of data and processing requirements, many organizations are considering possibilities to additionally optimize their infrastructures and collaborative processes by employing private cloud federations.
In this work, we present our contribution based on three real-world use cases implemented in the course of the SUNFISH project. We consider intra- and inter-organizational processes which demand secure and transparent infrastructure and data sharing. Based on derived requirements for data security and privacy in cloud federations, we propose a security governance architecture which enables a multi-layered, context and process-aware policy enforcement in heterogeneous environments. The proposed architecture relies on the micro-services paradigm to support scalability and provides additional security by integrating reactive and transformative security controls. To prove the feasibility of this work we provide performance evaluation of our implementation.

Keywords: Authorization, Cloud security, Cloud federation, API security, Data masking, Data security policy language, XACML

1 Introduction

The adoption of the cloud paradigm enabled public administrations to realize some of the benefits which come with it, including optimizations of investments as well as increased degrees of integration, scalability and flexibility [10,8]. However, due to the legal limitations [11] and concerns related to security, availability and reliability [22], many entities focused on the deployment of private clouds.

Two integration trends can be observed: firstly, the variable utilization of cloud infrastructures based on peaks often results in under-utilization on average, which leads to inefficiency of investments [5]. Secondly, there is an increasing need across public administrations to collaborate by exchanging data or taking

part in joint process workflows. This exchange is, however, constrained by data security and privacy concerns. Due to the lack of open infrastructures to ensure conformance to legislation and security requirements, public entities are hindered from sharing data and infrastructure with each other. Instead, alternative and suboptimal mechanisms have to be employed, increasing overall costs and overheads and negatively impacting the proliferation of innovative eGovernment services.

In this work, we present our contribution realized in the SUNFISH project, an initiative to enable secure federation of private clouds based on the requirements of the public sector. By addressing the needs of three use scenarios, this project aims to provide a framework for secure and efficient cloud federations which allow transparent data and service integrations in heterogeneous environments.

Realised under the broader project scope, the contribution of this work focuses on the security enforcement architecture of the SUNFISH framework. Our proposal provides an extension to the XACML approach [18] by introducing contextual and process awareness in collaborative policy management for federated environments. The proposed architecture relies on the micro-services paradigm to support scalability and provides additional security by integrating proactive and transformation-based security controls. To prove the feasibility of this work we evaluate performances and scalability of our implementation.

1.1 Paper Organization

This work is structured as follows. In Section 2 we describe the motivation and challenges for this work, introduce the use case scenario and provide a high-level project overview. In Section 3 we apply an intra-organizational perspective on agent-based transactions and establish building blocks for the framework's security architecture. In Section 4 we integrate these blocks and introduce the architecture for security governance. Section 5 provides the evaluation of the proposed architecture in the terms of imposed overhead. In Section 6 we describe related work and conclude this paper in the subsequent section.

2 Background

Various public administrations across Europe already utilize a variety of cloud computing concepts. However, due to regulatory limitations, most public entities only make use of private cloud setups. Typically, these private clouds are dimensioned to handle the public administration's peak loads within a reasonable time frame. The rest of the time such setups remain idle. Peak times often occur in a predictable manner, once per month or every few months, depending on the application. This leads to a situation where many clusters with massive computational power are available across Europe but partitioned and practically unusable for other public administrations.

Due to the growing demand on disposable computing resources, organizations across Europe recognized the advantages of extending the reach of private

clouds beyond their own boundaries. By taking part in technical and organizational processes related to the establishment of private cloud federations, these organizations can reap additional benefits resulting from synergistic effects. Instead of relying on proprietary solutions and complex processes to establish cross-organizational data sharing, public entities can employ existing federated infrastructure with its technical and governance controls to structure transparent, scalable and secure collaborative workflows. Moreover, as a result of the integration of accountable processes and reusable components provided by the framework, public entities may realize additional gains by decreasing overall overheads and barriers related to legal and security requirements.

2.1 Challenges

The overall challenge of the SUNFISH project is to allow for a secure federation of private clouds of public administrations, supporting a more efficient utilization of the available resources. SUNFISH does not aspire to provide a single cloud across Europe, but aims at providing a lightweight wrapper to be installed on top of any cloud infrastructure in order to become part of a federation. To achieve this goal, various obstacles need to be overcome:

- Adherence to different data security and protection rules of the federation participants in cross-border and intra-country transactions,
- Controlled data exchange, by revealing sensitive data up to a level as required by the processes,
- Integration of heterogeneous infrastructures of participating private clouds,
- Enabling the process integration based on different technological bases.

The aim of this solution is to enable a federation based on different scenarios. First, organizations may operate multiple data centers at various geographic locations, subjected to various legislations. Regulations may apply to prohibit the transfer of sensitive data out of a certain geographic region, making this scenario infeasible in particular cases. Another scenario is the exchange of data or sharing of infrastructure with other entities. This is a highly dynamic task in terms of characterizing transmitted data and deciding which parts are allowed to be transmitted in plain text. These constraints require a flexible framework for specifying policies on data and associated mandatory actions to be performed in terms of transparent data transformation.

2.2 Use Case Scenario

The SUNFISH framework evolved based on three use case scenarios that involve public administrations from three European countries. A detailed overview of these scenarios has been provided in [24]. Figure 1 shows a generalized use case which assumes that multiple organizations may join the federation for the purpose of sharing their resources or exchanging data. Participants may dedicate all or part of their resources, in terms of virtual machines, to the federation. More details on establishing a SUNFISH federation and related technical

building blocks are provided by Bottoni et al. [7]. Their architecture includes microservice-based management layer that enables automated horizontal scaling and dynamic workflow management of federation services based on variable load.

The federation manages available virtual machines and bootstraps by building a so-called *infrastructure tenant* responsible for management of common services, including the data security policy store, policy evaluation services, workload management or common monitoring infrastructure. Participants of the federation may deploy applications and services in the federation. Using a common administrative console, participants can define security policies that apply to resource management and data flows on different levels of abstraction. For a distinction between different types of entities, we refer to Section 3.

A typical scenario assumes an application deployed in *Tenant A* to use a service deployed in *Tenant B* to perform arbitrary data processing. Virtual machines in *Tenant B* might not be permitted to access sensitive data in plain. Therefore, the federation needs to assure that data transferred from *Tenant A* to *Tenant B* do not contain sensitive information by encrypting or masking relevant parts. Once transformed and transferred back from *Tenant B* to *Tenant A*, data may need to be decrypted or unmasked to continue the processing [25].

Applications can employ services deployed at fully trusted tenants or at special segregated tenants to perform calculations on sensitive data. Segregated tenants need to fulfil high standards in terms of data security and protection of their assets. The concrete implementation and security measures need to be agreed on between the participants of the federation.

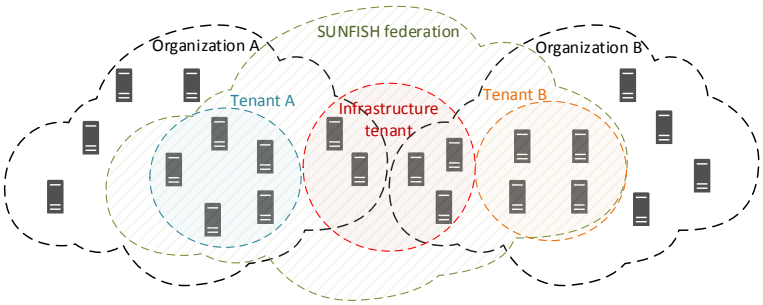

Fig. 1. Overview of our general use case of the SUNFISH federation

2.3 Contribution

This paper builds on and extends our previous work in the scope of SUNFISH project. In our initial contribution [25], we analyzed access control models, policy approaches and cryptographic building blocks applicable to federated environments. Our second contribution [23] discusses the emerging model of cloud-based integration platforms, where workflows dominantly get executed in the cloud. We proposed service and policy decision models for API-based interactions, supporting existing OAuth 2.0 scenarios. In the third relevant contribution, we introduced the overall concept of SUNFISH [24]. We provided a state-of-the-art

assessment and gap analysis in terms of data security, identifying gaps and providing a roadmap for further research. We furthermore established cross-tenant communication models, defined the framework for dynamic data transformation and introduced decentralized policy decision models. These contributions serve as a basis for the current work.

Our contribution in this paper includes an enhanced security policy model and the associated architectural framework for security governance in multi-organizational integrations. We particularly consider inter-cloud environments based on heterogeneous actors and service-oriented architectures. Based on that, we establish an interaction model that introduces additional granularity level, enabling enhanced context and process sensitivity in definition and enforcement of security policies. We apply this model to serve in a context of cross-entity transactions between (semi)autonomous agents in different roles. The architecture and proof-of-concept implementation presented in this work target a federated management and enforcement of policies on different federation layers. Finally, we evaluate the overhead and scalability of our implementation.

3 Modelling Inter-Cloud Interactions

In this section, we establish an abstract entity, interaction, authorization and administration models for security enforcement in collaborative private cloud federations. The graphical summary of our models is shown on Figure 3.

3.1 Entity Model

Based on the needs derived from the use cases, we identified two subjects in inter-cloud transactions. An *application* is a system or a component which stores and processes data by taking part in inter-organizational or intra-organizational processes. In a typical scenario, an application connects to a *service* hosted at an adjacent tenant, issuing a request to manage data or perform a process.

In our distinction between services and applications we follow the concepts established by Keen et al., referring to services as entities which expose functionalities and resources using explicit, implementation-independent interfaces, with the goal to be reused based on business needs of several entities [14]. As they support loose coupling and integration using separate communication protocols, services facilitate location transparency and interoperability [14,21].

Our distinction also considers that services may expose resources on a granular level, providing parts of application systems which relate to particular problems or domains [15]. While services are meant to run continuously, interacting with applications or other agents, applications may be controlled by the user, with the purpose of accomplishing a particular process which may span across different domains and execute for a limited period of time.

3.2 Interaction Model

The transactions occurring between applications and services at different tenants can be described using various levels of abstraction. Due to the relevance for data

protection and overall security, we establish a transactional model which enables granular and contextual referencing of interactions in their various cycles.

In our model, we consider an *interaction* as a transactional unit which consists of *request* and *response* cycles. Furthermore, we consider a *transaction* as a set of entity interactions related to a context. Based on that, we identify three common types of transactions which occur during service-based, cross-tenant interaction flows. These types include *synchronous*, *synchronous with polling* and *asynchronous* transactions, as depicted in Fig. 2. Notations using θ and φ in the figure refer to request and response cycles, with numbers specifying the interaction sequence, and *full* and *part* describing the extent of the provided data.

In the first case (a) shown in Figure 2, the overall flow consists of one request-response interaction cycle, where the application provides the full data set to the service, which responds with the corresponding data set after processing. This workflow is referred to as *synchronous* interaction.

The second case (b) shows a transaction consisting of two flows. First, the application provides the request with the data to be processed, receiving a confirmation from the service with a reference to and the status of the request. Later, during the n-th interaction, the application contacts the service, checking the request status or trying to retrieve the processing result. We refer to this approach as *synchronous with polling*. The subsequent requests contain the reference to the original request, while the service responds either with the status description (part) or a full data set. In the latter case, the transaction can be considered as completed, after the processed data has been sent back to the requester.

The third case (c) illustrates an *asynchronous* interaction which includes similar checks as (b), but using callbacks. Instead of polling the status of a request and eventually retrieving the result, the requester relies on a callback, a response which is initiated by the service in the second flow.

Based on these descriptions we distinguish between synchronous and asynchronous flows. Synchronous transactions do not require a significant amount of processing time and may be completed in a relatively short period of time. On the other hand, transactions which require additional time to complete or include huge batches of data to be processed may be organized in an asynchronous manner, consisting of several interactions (request-response cycles).

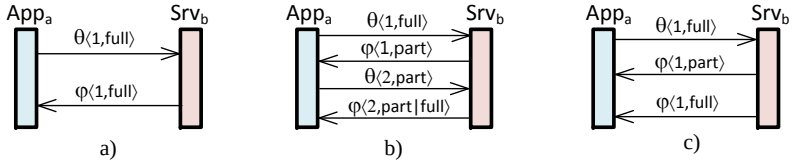

Fig. 2. Interaction model and flows for resource-aware integrations

The characterization of these transactions is relevant for the definition of security policies and processing requirements, as it enables evaluating the interactions occurring in different contexts and for different purposes. This furthermore allows grouping several interactions which are a part of one process, assigning them to a particular security context. Security enforcement components in the SUNFISH framework rely on underlying processes and flows, aiming at achieving

optimal security without imposing additional overhead on the application layer or altering the underlying interactions and process flows. Hence, by applying this model we establish security components as process-aware and are able to maintain a holistic view of communication flows.

3.3 Policy Specification

To establish an access control framework in the federation, we apply our refined attribute-based model for federated collaborative environments. While other models, such as ABAC for web services by Yuan and Tong [26], or the proposals by Jin et al. [13] and Ardagna et al. [4] employ entities such as *subjects*, *resources*, *environments* or *objects*, we apply a more general approach which considers agent-based interactions.

In our refinement we observe an *access* as a sequence of lower-level activities which may produce different data security effects at each step. For this reason we model any involved *entites*, *processes* and *data* to describe a particular event or authorization in a granular and context-sensitive manner. We furthermore integrate the concept of *obligations*, which represents a process which has to be satisfied during the interaction flow. In contrast to *ABC Core* by Park et al. [19], obligations in our model employ other functional components of the framework, being executed and verified by a trusted architecture.

Based on the identified entities and interactions, we model authorization using the following concepts, as illustrated in Figure 3:

- *Service* describes the target service, its data and behavioral characteristics.
- *Application* models the origin application and its environment.
- *Request* relates to the first interaction step, modeling provided data, access semantics, expected constraints and enforced transformations.
- *Response* is analogous to the request category, characterizing the second interaction step related to the response of a target service.
- *Obligation* models necessary processes which have to be executed prior to completing interaction at each level of granularity.

Although not relevant to reach an authorization decision, we consider obligations as a part of the whole interaction process and a necessary step which has to be enforced by the trusted infrastructure.

3.4 Policy Management

The subject in the proposed framework refers to one of the following entities:

- *Organization*: a basic organizational unit and federation member, typically a legal entity which takes part in the federation processes.
- *Federation*: entities governing the federation in a global administrative role.
- *Tenant*: a logical unit which comprises resources provided to the federation.

From a functional point of view, a tenant relates to a *security compartment*, [20,2] serving as a basic unit subject to security management in the framework [6]. Organizationally, tenants encompass resources provided by the originating organization, representing its logical unit, distinctive according to its functional, organizational or geographical characteristics. Tenants which include resources from multiple federation members are considered logical constituents of the federation, dedicated to providing common federation resources and operations. These include administrative facilities, security governance infrastructure, monitoring infrastructure, identity management or workload scheduling components.

Following that, we establish policy management in three vertical planes:

- *Inter-cloud* policy management enables federation members in the administrative role to manage the security on the federation level, using high-level constructs to describe policies and processes.
- *Organizational* policy management enables the administrators of organizational members to specify security policies which address their resources and security needs.
- *Tenant-level policy management* enables administrators on the level of organizational sub-units to manage policies which relate to resources and processes from their domain.

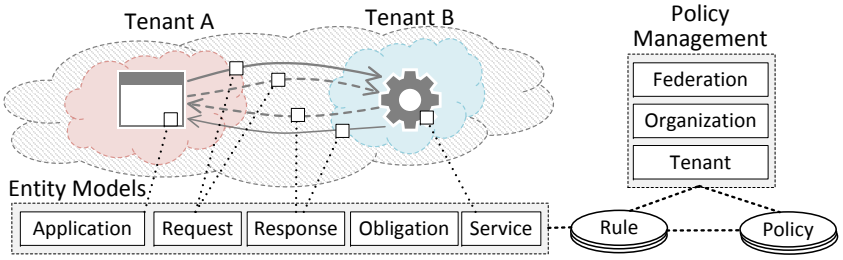

Fig. 3. Overview of authorization and management models for private cloud federations

4 Architecture

The *eXtensible Access Control Markup Language* (XACML) [18] is a de-facto standard in the field of data security policy languages. Traditionally, enterprises are managing their policies at the enforcement points directly. With the introduction of XACML a decoupled model was introduced, comprising different entities with well-defined responsibilities.

The core components of the enforcement model as defined by XACML include the *Policy Enforcement Point* (PEP), the *Policy Decision Point* (PDP), the *Policy Administration Point* (PAP) and the *Policy Information Point* (PIP). The responsibilities of these components are clearly assigned. The PEP, as a contact point for applications, issues requests to the PDP and enforces the received decisions. The PDP may contact various PIPs to gather additional attributes, and contacts the PAP (also referred to as *Policy Retrieval Point* (PRP)) to collect relevant policies. The policies are then evaluated by the PDP and a decision is returned to the PEP.

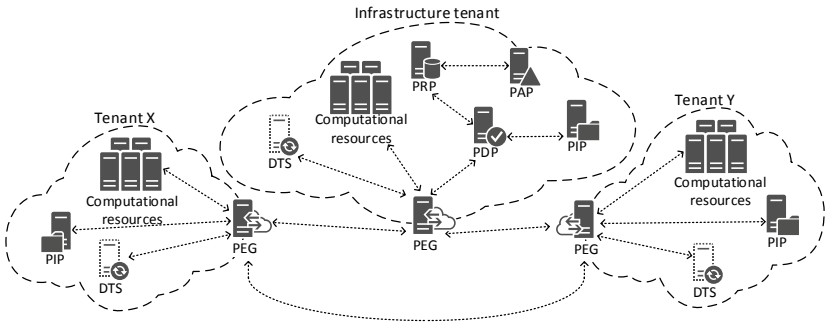

Fig. 4. Overview of the proposed architecture

Our proposed architecture provides a concrete implementation of the *XACML-approach* for federated cloud environments, and specifies the missing interactions, interfaces, and workflows.

The proposed enforcement architecture is illustrated in Figure 4. It is based on the structure and data-flow requirements identified in the general use case from Section 2.2. Tenants represent security compartments as a logical construct to group workflows and data-flows with the same requirements on data security. The special infrastructure tenant is used to deploy common services or management facilities like policy decision services and policy stores. All compartments operate with SUNFISH dedicated computational resources (e.g. virtual machines), which might be used by application or services.

The PDP adheres to the OASIS XACML specification as close as possible. We define that the policy decision process fails if attributes referenced in policy are missing in the decision request and cannot be retrieved using one of the available PIPs. Moreover, we introduce a clear distinction between PRP and PAP. The PAP is an administration tool to manage policies in the policy store. The policy store can be accessed through the PRP to retrieve corresponding policies or add, remove or modify policies (depending on the granted user rights).

The core component of the enforcement infrastructure is the *Policy Enforcement Gateway* (PEG). It governs and safeguards all outgoing and incoming data flows of one particular security compartment. The component is not included in the XACML specification but has a close relationship to the PEP. In fact the PEG instantiates the PEP and adds gateway functionality.

To adhere to data security and data protection requirements, sensitive data might be cryptographically protected before leaving a security compartment. Cryptographic operations are in control of the *Data Transformation Service* (DTS), which may offer operations like (order-preserving, format-preserving) encryption, data masking or anonymization [25]. The execution of these operations is defined in the scope of security policies and managed by the compartmental PEG in the form of *obligations*, as described in Section 3.4.

The remainder of this section describes our enhancements to the existing components and the novel PEG in detail.

4.1 Policy Retrieval and Administration Point

Policy retrieval is a critical workflow in policy-based systems, inherently impacting the performance of the whole system. In an optimal scenario, the PRP only returns relevant policies to the PDP, in the worst case the PRP is just a basic policy repository returning all available policies.

XACML defines so-called *targets* which indicate whether a policy applies to a request or not. These targets represent matching conditions as the disjunctive normal form (DNF) of an arbitrary set of matching functions defined in XACML. In essence, functions evaluate to ternary literals (`MATCH`, `NOMATCH`, `INDETERMINATE` based on the input request) as part of the conjunctions of the DNF-representation of the target. A target evaluation result of `INDETERMINATE` at the PRP means that not all required attributes are available and the policy is included in the policy result-set.

Further improvements were made in terms of policy storage. XACML enables a modular definition of policies in a re-usable manner. We introduce the concept of *root policies* and *re-usable policies*. In the process of determining matching policies, only root policies are considered in the first place. Root policies then may reference re-usable policies. This keeps root policies clearly arranged and improves the performance of determining relevant policies. Administrators are still able to structure and re-use their policies.

The responsibilities of the PAP, in contrast to the XACML specification, were streamlined to act as a point of structured management for policies. Administrators require adequate permissions to access the PAP and might only be allowed to add policies for specific areas.

4.2 Policy Enforcement Gateway

According to the XACML specification the PEP's responsibilities are to contact the PDP to acquire and enforce a policy decision. Following this approach applications actively need to communicate with or integrate the PEP. This way many responsibilities are shifted to the application. We extend this model by introducing the *Policy Enforcement Gateway* (PEG). It is deployed on the edge of each security compartment and governs the whole communication passing the compartment borders. No communication bypassing the PEG is possible.

Conceptually, the PEG is a PEP extended with gateway functionality. The incoming requests are converted to XACML decision requests with predefined attributes as described in Section 4.3. The PEG then enforces the decision (a) by forwarding the request to the respective service or (b) by denying access to the service. Furthermore, obligations are automatically fulfilled in the process of enforcing a decision. At the receiving end the target-zone PEG is invoked to check if policies allow access to the service. Finally the PEG of the target zone invokes the service and returns the result to the origin PEG, which passes the result back to the initiating application.

Using this approach we devised two ways to develop applications for this infrastructure: *SUNFISH-aware applications* are actively aware of the enforcement

infrastructure and directly contact the PEG to initiate a service call. These applications are comparable to applications written for unmodified XACML stacks, but using the advanced SUNFISH interface. *Transparent applications* are not aware of the enforcement infrastructure, accessing the services in a standard manner. In fact, their calls are intercepted by the PEG and are routed through the enforcement infrastructure.

For transparent applications to work, the PEGs need to be aware of the used protocol and they need to provide a converter from the application protocol to the inter-PEG protocol and vice-versa. Another burden to overcome regarding transparent applications is the usage of SSL/TLS. Technically, the PEG is a man-in-the-middle and one goal of SSL/TLS is to prohibit men-in-the-middle. Solutions like *mitmproxy*[1] exist for HTTPS, but require access to an application's trust store.

The PEG offers two REST endpoints for the application and for other PEGs following a similar schema. Requests to these endpoints encapsulate the original service request or response and add additional data:

- Original header data, used method and path are consolidated in a custom *RequestData* header.
- The service is identified by an id encapsulated in a custom *Service* header; no host or path to the service is transmitted. An enhanced version of the PIP acts as a registry to retrieve host, path and protocol of registered services.

Using this data, the PEG generates an XACML decision request, which is issued to the PDP with attributes as discussed in Section 4.3.

4.3 Attribute Groups

XACML is an open and extensible system where attributes can be defined without limitations. This makes it impossible to design efficient systems, and it may be one reason why the XACML specification does not tackle performance issues like fast and efficient retrieval of policies or policy indexing. We restrict valid attributes to a specific set to optimally meet the use case of federated cloud computing. We introduce four categories to be used in our use case:

- *Service-related attributes* contain information about the target service like: id, host, port, and zone. Services need to be registered in the federation and can be referenced by id.
- *Application-related attributes* contain information about the origin application similar to the service-related attributes.
- *Request-related attributes* contain information regarding the current request like: protocol, content-type, header parameters, url parameters, or body data. In this case header, url and body data are not referring to the real data, but to a characterization of this data. In the simplest case this could just be the content type. In a more complex scenario the operators may register an XSD at the PIP, and the PEG could then check if the body data

[1] https://mitmproxy.org/

adheres to the specified XSD. Using this approach, policies can be defined based on the type of transferred data.
- *Response-related attributes* are defined analogues to request-related attributes, but characterize the response of a target service.

5 Deployment and Evaluation

In order to evaluate the performance overhead of the enforcement infrastructure, a basic scenario is being evaluated. The turnaround time of a request submitted to a service guarded by an enforcement infrastructure is compared to the turnaround time of a request directly issued to the same service. This service does not perform any computationally intensive tasks in order to process the request. Instead, predefined responses are being served. By trying to keep the impact of the service on the overall turnaround time to a minimum, the overhead caused by the enforcement infrastructure is highlighted.

5.1 Benchmarking Setup

All benchmarks are executed on an Amazon *EC2 m4.xlarge*[2] [1] instance running the 64 bit version of *Ubuntu 14.04 LTS*. The use of an EC2 instance makes it possible to directly translate the time overhead to additional costs introduced by the enforcement infrastructure. The setup consists of the following components:

- The *service* which delivers structured data
- A single *policy enforcement gateway*, which controls information flow
- One PIP used by the PEG to enquire information about the requested service
- A PDP, which evaluates authorization requests
- One PRP providing the PDP with policies matching the incoming request
- Another PIP providing the PDP with missing attributes

All components of this test setup are deployed on the same host to eliminate network latency from the measurements. As can be seen in Figure 5, the enforcement infrastructure requires nine additional remote calls in order to serve a request. We provide a description of these workflow steps in Listing 1.
The application issuing the requests directly conducts the measurements. This way any potential overhead resulting from, for example, OS-level management is eliminated. Each request is issued 100 times, the median turnaround time and the interquartile range (IQR) are evaluated. The bootstrap time for the setup and loading of policies is neglected by omitting the first request-response cycle from the measurements.

The dynamic nature of the approach may cause greatly varying turnaround times, depending on factors such as the complexity of a request or the number of policies managed by the PRP. Therefore, different configurations are evaluated to test and cover a range of representative variations of the same setup.

[2] The *EC2 m4.xlarge* features 4 vCPUs, 13 *EC2 Compute Units* and 16 GiB of RAM

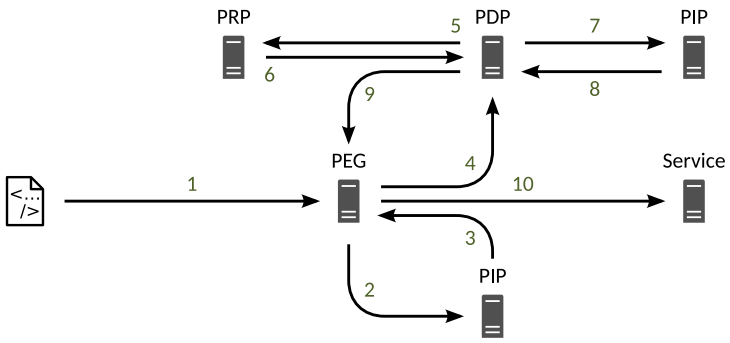

Fig. 5. Workflow of the setup used for benchmarking

1. The original request is submitted to the PEG.
2. The PEG processes the request headers and queries its PIP for the service location.
3. The PIP returns the location of the service associated with the provided identifier.
4. The PEG generates and provides the authorization request to the PDP.
5. The PDP queries the PRP for matching policies.
6. The PRP matches, retrieves and returns the set of matching policies to the PDP.
7. The PDP forwards the request to PIP to get additional attributes for a decision.
8. The PIP enhances the received request with the required attributes and returns it.
9. Given a sufficiently populated request, the PDP reaches a decision, which is forwarded to the PEG.
10. The PEG enforces obligations imposed by the decision and issues the (transformed) request to the service.
11. Depending on the obligations, additional entities (such as a masking service) may be involved (not part of the evaluated scenarios).

Listing 1. Evaluating requests in the security governance architecture

Each of two following subsections is dedicated to one setup configuration. These configurations are benchmarked starting with a PRP containing only a single matching policy, with subsequent instances featuring an increasing number of root policies, topping out at 1000. The nature of the policies (and their targets in particular) are described for each configuration individually. In general, however, the PRP and the PDP are stressed the most as these components provide the most computationally intensive services. Finally, the results obtained from each configuration are compared to each other in order to deduce a correlation between the number of policies, the number of applicable policies, their complexity and the overall turnaround time of the enforcement infrastructure.

5.2 Configuration 1: All Policies Match Any Request

The initial configuration features a PRP with a set of root policies matching any request using an empty-clause target. This setup equally stresses the PDP and the PRP, since the whole repository of policies is delivered to the PDP for evaluation, regardless of a request. The PDP then has to re-evaluate the whole policy set to reach an authorization decision. The reason for a full re-evaluation

is the possibility of indeterminate matching outcomes. In such cases the PDP has to contact its PIP(s) to enhance the request with additional attributes, in order to determine the actual set of matching policies. While this process may seem gravely inefficient, the current configuration represents the absolute worst-case scenario regarding the number of matching policies. Realistically, the PRP will only deliver a fraction of all the policies in its repository for each request.

Table 1 summarizes the turnaround times for a request issued to a service guarded by the enforcement infrastructure. As can be seen, the introduced overhead is significant and substantially increases with the number of policies.

	Direct	1	10	100	1000
3rd quartile	2	111	139.5	402.5	1927.5
Median	2	104	131.0	355.0	1909.0
1st quartile	2	98	118.0	331.5	1885.5

Table 1. Turnaround time (in ms) of Configuration 1 based on the number of policies

Figure 6 illustrates the characteristics of the overhead shown in Table 1. The turnaround time for the unguarded service (labeled as *Direct*) is included for comparison. Obviously, deploying the enforcement infrastructure causes a considerable time overhead. When only considering root policies featuring empty targets, however, the presented figures lead to a wrong conclusion.

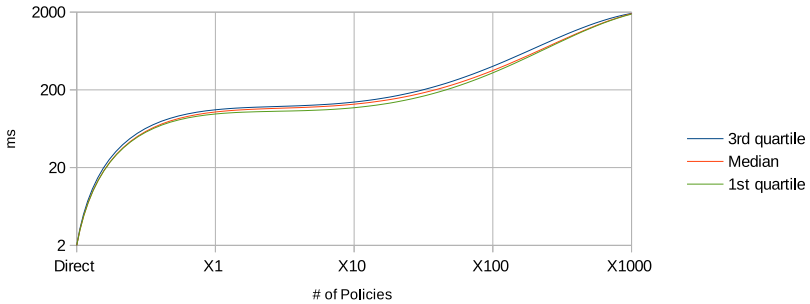

Fig. 6. Plot showing the impact of Configuration 1

5.3 Configuration 2: Complex Targets, 50% Matching

The second configuration introduces a set of root policies featuring complex targets. This results in roughly 50% of all policies matching a request. As can be seen in Table 2 and Fig. 7, the turnaround times are significantly lower compared to Conf. 1. Matching and the evaluation performed at the PDP, however, are a few orders of magnitude more complex compared to the first configuration.

	Direct	1	10	100	1000
3rd quartile	2	100.5	102.5	147.5	462.5
Median	2	95.0	96.0	121.0	433.0
1st quartile	2	88.5	90.0	100.0	418.0

Table 2. Turnaround time (in ms) of Configuration 2 based on the number of policies

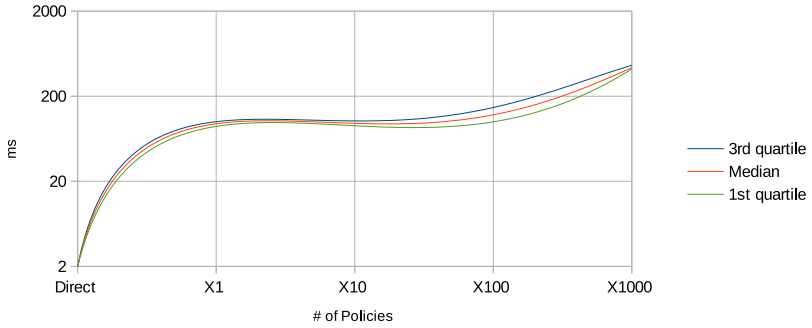

Fig. 7. Plot showing the impact of Configuration 2

5.4 Discussion

The reason for the decreased turnaround times in Conf. 2 is the lower amount of data transmitted between the PRP and the PDP. In the current proof-of-concept implementation of the enforcement infrastructure, the (de-)serialization process is to blame for this delay, as it is not optimized for speed, but for simplicity and debuggability. Benchmarks conducted by Maeda [16] suggest that it is possible to streamline this process by carefully choosing a serialization library suited to the use case at hand. Considering these results, it is expected that an optimization in this area will lead to the matching and decision processes being the dominant factor of the introduced overhead. Consequently, the actual business logic scales exceptionally well with the number of root policies.

Conf. 2 is, in fact, one of the worst possible cases concerning the matching targets defined in policies. Applying simple arithmetic further points to the conclusion that the overall impact of an increasing number of policies is negligible. When considering a real-world service which performs computationally intensive tasks on large amounts of data, the service and partially the network latency are expected to become the dominant factor of the total turnaround time. The overhead introduced by the enforcement infrastructure as a whole is therefore considered negligible compared to network latency and other sources of jitter.

The above statements also hold true even for a considerably larger amount of overall policies, since the matching speed is only affected by the number and complexity of root policies. Typically, such complex setups will rely on reusable policies referenced within root policies and root policy sets. No matching has to be performed for re-usable policies, which makes these ideal candidates for caching, completely eliminating any overhead introduced by (de-)serialization. The outcome of matching processes themselves can also be cached, however, the impact of such an extended caching strategy requires further investigation.

In conclusion, the introduced overhead of a few hundred milliseconds must not be considered a slowdown by two orders of magnitude, but rather a constant overhead. Considering the use case of agent-based interactions in federated cloud

environment, even an enforcement infrastructure introducing a delay of few seconds is acceptable.

6 Related Work

Policy languages enable data owners to specify access controls and usage controls for their resources. Specifying and enforcing policies is still subject to research. Multiple other proposals are available, using approaches similar and different to XACML: The Formal Access Control Policy Language (FACPL)[3] features a more lightweight syntax to express access control policies. Damianou et al. [9] introduce the Ponder policy language, providing a more fine-grained approach through a categorization of access-control policies into: authorization policies, information filtering policies, delegation policies, and refrain policies. Hilty et al. [12] also focus on the data usage aspect: who may access data, what actions are allowed to be performed on the data. All these approaches are either theoretical concepts or barely used in practice. The only de-facto standard being broadly employed in the industry, is (XACML) [18].

Multi-tenant access control for intercloud proposed in recent work of Ngo et al. [17] relies on infrastructure description models to generate policies for dynamic objects from predefined policy templates. Their model deals with the policies on infrastructural level and does not assume collaborative and distributed policy management.

On an European level, multiple projects have been performed to reach similar goals but using other approaches or providing building blocks which can be used in our solution: PRISMACloud[4] is in the process of developing next generation cryptographically secured services in the cloud, which might be used by our solution. In the PrimeLife[5] project, extensions to the XACML policy language were developed to better enable the data controller to specify access restrictions on certain resources [3]. The TClouds[6] project focuses on delivering computing and storage resources with a new level of security, privacy, and resilience. This goal is achieved by establishing trust to remote cloud computing resources by utilizing certain building blocks like: access-control-as-a-service, cryptography-as-a-service, secure-logging.

7 Conclusions and Future Work

Although the cloudification enabled public entities to consolidate and optimize their infrastructures, in the current setup the effects of this process are limited due to peak-driven usage patterns. The next evolutionary step would be the horizontal integration of public entities by federating their private clouds. However,

[3] http://rap.dsi.unifi.it/facpl/guide/FACPL-guide.pdf
[4] https://prismacloud.eu/
[5] http://primelife.ercim.eu/
[6] http://www.tclouds-project.eu

due to lack of supporting frameworks, the legal constraints and security concerns are prevailing. Many entities hence restrain from sharing their resources, resulting in a suboptimal utilization of existing setups in overall.

The SUNFISH project aims to address these issues by enabling secure federation of private clouds of public administrations. The goal of the project is to advance current state-of-the-art by providing an open framework that conforms to high security and privacy requirements of public administrations and supports a diverse range of systems.

Building on our previous work, in this paper we presented an enhanced security policy model and associated architectural framework that enable security governance in multi-organizational service integrations. In our approach, we considered use case of interactions between (semi)autonomous agents, situated in different domains. We introduced a perspective that enables granular modeling of their interactions, enabling context and process awareness in definition and enforcement of security policies. Based on a proof-of-concept implementation, we demonstrated scalability and feasibility of our proposal.

In the next phase of our work we intend to integrate existing, proactive security enforcement with reactive security controls based on monitoring and continuous anomaly detection. With this setup we aim to provide a holistic solution for security enforcement in a multiorganizational environment, providing public entities with additional accountability-supporting mechanisms.

Acknowledgments

This work has been supported by the EU H2020 Programme under the SUNFISH project, grant agreement N.644666.

References

1. Amazon Web Services, Inc.: Amazon EC2 Pricing (2016), https://aws.amazon.com/ec2/pricing/
2. Archer, J., Cullinane, D., Puhlmann, N., Boehme, A., Kurtz, P.: Security guidance for critical areas of focus in cloud computing v3.0. Cloud Security Alliance (2011)
3. Ardagna, C.A., di Vimercati, S.D.C., Neven, G., Paraboschi, S., Preiss, F.S., Samarati, P., Verdicchio, M.: Enabling privacy-preserving credential-based access control with XACML and SAML. In: Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on. pp. 1090–1095. IEEE (2010)
4. Ardagna, C.A., Cremonini, M., De Capitani di Vimercati, S., Samarati, P.: A privacy-aware access control system. Journal of Computer Security 16(4) (2008)
5. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (2010)
6. Bacon, J., Eyers, D., Pasquier, T.F.M., Singh, J., Papagiannis, I., Pietzuch, P.: Information flow control for secure cloud computing. IEEE Transactions on Network and Service Management 11(1), 76–89 (2014)

7. Bottoni, P., Gabrielli, E., Gualandi, G., Mancini, L., Stolfi, F.: Fedup! Cloud federation as a service. European Conference on Service-Oriented and Cloud Computing (2016)
8. Carroll, M., Van Der Merwe, A., Kotze, P.: Secure cloud computing: Benefits, risks and controls. In: 2011 Information Security for South Africa. pp. 1–9. IEEE (2011)
9. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy specification language. Proceedings of Policy 2001: Workshop on Policies for Distributed Systems and Networks pp. 18–39 (2001)
10. De Chaves, S.A., Uriarte, R.B., Westphall, C.B.: Toward an architecture for monitoring private clouds. IEEE Communications Magazine 49(12), 130–137 (2011)
11. Deussen, P., Eckert, K.P., Strick, L., Witaszek, D.: Cloud concepts for the public sector in Germany – use cases. A publication by Fraunhofer Institute For Open Communication Systems (2012)
12. Hilty, M., Pretschner, A., Basin, D., Schaefer, C., Walter, T.: A policy language for distributed usage control pp. 531–546 (2007)
13. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering dac, mac and rbac. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 41–55. Springer (2012)
14. Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., Verschueren, P.: Patterns: Implementing an SOA using an enterprise service bus. IBM Redbooks 336 (2004)
15. Kohlborn, T., Korthaus, A., Chan, T., Rosemann, M.: Identification and analysis of business and software services - a consolidated approach. IEEE Transactions on Services Computing 2(1), 50–64 (2009)
16. Maeda, K.: Performance evaluation of object serialization libraries in XML, JSON and binary formats. In: Digital Information and Communication Technology and it's Applications, Second International Conference on. pp. 177–182 (2012)
17. Ngo, C., Demchenko, Y., de Laat, C.: Multi-tenant attribute-based access control for cloud infrastructure services. Journal of Information Security and Applications 27, 65–84 (2016)
18. Parducci, B., Lockhart, H.: eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard (January), 1–154 (2013)
19. Park, J., Sandhu, R.: The UCON ABC usage control model. ACM Transactions on Information and System Security (TISSEC) 7(1), 128–174 (2004)
20. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proceedings of the IEEE 63(9), 1278–1308 (1975)
21. Schmidt, M.T., Hutchison, B., Lambros, P., Phippen, R.: The enterprise service bus: making service-oriented architecture real. IBM Systems Journal 44(4), 781–797 (2005)
22. Shin, D.H.: User centric cloud service model in public sectors: Policy implications of cloud services. Government Information Quarterly 30(2), 194–203 (2013)
23. Suzic, B.: User-centered security management of API-based data integration workflows. In: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium (2016)
24. Suzic, B., Reiter, A.: Towards secure collaboration in federated cloud environments. In: 11th International Conference on Availability, Reliability and Security (ARES) (2016)
25. Suzic, B., Reiter, A., Reimair, F., Venturi, D., Kubo, B.: Secure data sharing and processing in heterogeneous clouds. Procedia Computer Science 68(316) (2015)
26. Yuan, E., Tong, J.: Attributed based access control (ABAC) for web services. In: IEEE International Conference on Web Services (ICWS'05). IEEE (2005)