A Framework and Risk Assessment Approaches for Risk-based Access Control in the Cloud

Daniel Ricardo dos Santos[a,*], Roberto Marinho[a], Gustavo Roecker Schmitt[a], Carla Merkle Westphall[a], Carlos Becker Westphall[a]

[a]*Networks and Management Laboratory*
Department of Informatics and Statistics
Federal University of Santa Catarina
88040-970 - Florianópolis - SC - Brazil

Abstract

Cloud computing is advantageous for customers and service providers. However, it has specific security requirements that are not captured by traditional access control models, e.g., secure information sharing in dynamic and collaborative environments. Risk-based access control models try to overcome these limitations, but while there are well-known enforcement mechanisms for traditional access control, this is not the case for risk-based policies. In this paper, we motivate the use of risk-based access control in the cloud and present a framework for enforcing risk-based policies that is based on an extension of XACML. We also instantiate this framework using a new ontology-based risk assessment approach, as well as other models from related work, and present experimental results of the implementation of our work.

Keywords: access control, cloud computing, risk

1. Introduction

Cloud computing enables the delivery of computational resources and services through the Internet, providing easy access, elasticity and resource sharing [1]. The cloud model is widely adopted because of its economical and performance advantages for customers and service providers. However, the growing number of users and available resources, as well as the diversity of supported applications, emphasize the security challenges of this model [2].

Access control is crucial to ensure the correct enforcement of security policies on the cloud. There are well-known solutions to enforce policies based on traditional access control models, such as the eXtensible Access Control Markup Language (XACML) [3]. Nonetheless, the emergence of new requirements in access control, derived from current information security needs and the needs

*Corresponding author
Email address: `danielrs@inf.ufsc.br` (Daniel Ricardo dos Santos)

of highly dynamic environments, has led to the development of access control models based on risk assessment [4], for which clear enforcement mechanisms are not readily available. One of the main advantages of risk-based access control models is the ability to handle exceptional access requests, when a user must be granted access to perform a critical action, even though he or she may not be previously authorized to do so. Another issue solved by this kind of access control model is flexibility in accessing resources. Traditional models employ rigid and static access control policies that are not well suited to dynamic and heterogeneous environments like the cloud [5, 6], since those environments present a continuous change in the available users and resources and greater administrative complexity.

This paper presents a framework for dynamic risk-based access control for cloud computing. The system manages user access to cloud resources by quantifying and aggregating risk metrics defined in risk policies created by resource owners. The risk-based model is built on top of XACML and allows the use of, e.g., Role-based Access Control (RBAC) or Attribute-based Access Control (ABAC) coupled with risk analysis. This combination provides flexible access control for both users and Cloud Service Providers (CSPs).

We also present instantiations of our framework using diverse risk-based models. One instantiation is based on ontologies, which provides a formal model for the inference of contextual information in risk analysis. The use of ontologies for access control models to provide flexibility and dynamism in decision making has been exploited in some works [7, 8]. However, the use of ontologies in the context of dynamic risk assessment is a novel contribution.

This paper is the consolidation of some of our previously published research in risk-based access control for cloud computing [5, 9, 10] with the addition of new material and results related to instantiations of our framework. Section 2 introduces the main concepts related to cloud computing and access control. Section 3 presents our **main contribution**, the development of an extensible framework for risk assessment approaches for access control in cloud computing. Sections 4 and 5 present **other contributions**, namely the instantiation of the framework with risk-based models and the development of an ontology-based risk quantification approach for such models. Section 6 presents the use of the framework in the emerging scenario of cloud federations. Section 7 describes our implementation and experimental results. Section 8 discusses related work and Section 9 concludes the paper.

2. Background

Cloud computing allows access to a shared pool of configurable computing resources with five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service [1]. Despite the advantages of clouds, security is the main fear of potential users, especially in public deployments [2].

2.1. Risk-based Access Control

Access control ensures that access requests from users to objects are validated according to predefined rules [11]. These rules form an authorization policy and the way of defining and enforcing them constitutes an access control model. Traditional access control models are Discretionary Access Control (DAC), where the owner of a resource decides who can access it and how; Mandatory Access Control (MAC), where the policy is defined by the system; and RBAC, where subjects are grouped in roles to which permissions are assigned. DAC and RBAC are still the most widely used, however new system architectures and requirements have led to the development of models such as ABAC [12] and risk-based access control.

Attacks such as unauthorized disclosure of information, denial of service and information tampering are still critical. However, new kinds of systems, e.g., the Internet of Things (IoT), grids, and clouds, characterized by distribution, automatic reconfiguration and dynamicity of users and resources, present new challenges to access control models [13, 14, 15]. Three main problems with traditional access controls have been identified in the literature [16]:

- they are too rigid to handle exceptional situations in which the policy itself should be overridden in order not to stop the system;
- they do not meet the requirements of dynamic secure information and permission sharing in collaborative environments; and,
- they are not flexible enough to handle the changing behavior of users.

Risk-based access control is still in an initial phase of adoption, mostly in academic or experimental settings, but there is a growing need to define formal models and standard enforcement mechanisms for it. The idea behind risk-based access control is that each request must be analyzed dynamically, taking into account not only predefined policies but also contextual information such as operational risk, user need, and the benefit of an action [17, 18, 19, 20]. In real applications, many unexpected situations require policy violations, which may occur because policies are incomplete or conflicting. Two recurring examples for policy overriding are medical and military environments where, e.g., a nurse or a soldier must obtain restricted information to save a patient's life or to complete a mission. Policy overriding in these situations is known as "break the glass" and it prevents system stagnation [21]. Using a risk-based access control model, these requests may be exceptionally granted if the risk is acceptable.

Risk-based access control models are characterized by the use of a function that attributes a risk value to each request, and this function is the main difference between models. Risk estimation, taking as input several factors, is used to determine whether an access request should be granted or denied. The output of such a function is based on a risk threshold, and access is granted if the quantified risk for the access request is below this threshold, as in formula (1):

$$canAccess(s, o, a, c) = \begin{cases} 1 & \text{if } risk(s, o, a, c) < riskThreshold \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where s is a subject, o is an object, a is an action, and c represents contextual information. $risk(s, o, a, c)$ represents the risk of the subject performing the action on the object, given a context. A result of 1 represents granted access, whereas a result of 0 represents denied access. The dynamic nature of access control is captured in these models because access decisions vary according to the contextual information available at the time of request. Granting an access request may entail some form of monitoring after the decision, such as auditing, the fulfillment of obligations [22] (post-conditions that must hold for a user to keep access) or a reputation system [23].

The Risk-Adaptive Access Control (RAdAC) model [24] proposes the use of risk-based access control for military applications. The original work does not go into details about risk estimation, but it uses the concepts of operational need and security risk, with access being granted when the need is greater than the risk. Britton and Brown [25] present a quantification method for RAdAC based on expert opinion, wherein a list of risk factors is compiled and a value is attributed to each factor. Afterwards, weights are attributed to each value and the final result is the combination of all the factors and their weights.

There are several other risk-based models. Cheng et al. [26] and Ni et al. [27] propose the use of fuzzy logic to quantify risk, whereas Molloy et al. [28] suggest the use of classifiers trained with the history of access decisions. Shaikh et al. [23] show two methods considering the history of users and Wang and Jin [29] show an application for privacy-preserving health care systems. The biggest challenge for risk quantification is the uncertainty of information. Many works present ways of bypassing this uncertainty, using techniques based on, e.g., probabilistic inference, machine learning and decision theory. This suggests that there are many ways of approaching the issue, each with its advantages and disadvantages.

2.2. eXtensible Access Control Markup Language (XACML)

With the increasing complexity of access control systems, it is necessary to define reference architectures for authorization protocols, especially for distributed systems. XACML is a language for general purpose access control that supports policies, access requests, and responses in XML [3]. The standard is used in several industrial and academic applications and defines an architecture with the following components:

Policy Administration Point (PAP): retrieves policies that will be used by the PDP;

Policy Information Point (PIP): retrieves information that will be matched against the policy to reach the access decision;

Policy Decision Point (PDP): reaches an access decision, based on the retrieved information and policies; and

Policy Enforcement Point (PEP): the point of access of the user to the system, which protects a sensitive resource, receives access requests and sends them to the PDP.

The root of an XACML document is a *Policy* or a *PolicySet*, which can contain other policies or policy sets. A policy is expressed with a set of *Rules*, which are evaluated individually. The result of each evaluation by the PDP can be *Permit, Deny, Indeterminate*—when there is an error or a missing attribute—or *NotApplicable*—when the request cannot be processed by the system. Since a policy set can contain multiple policies, a policy can contain multiple rules and all can have different access decisions, there exist combination algorithms to group decisions and reach a final result.

2.3. Ontologies

An ontology is a machine readable specification of concepts, relations, functions, and properties of an abstract model of a real world phenomenon [30]. Ontologies are used to share and reason about information of a particular domain. The Resource Description Framework (RDF) provides a data model for semantic web annotations in the form of triples written as *(Subject, Property, Object)*. The RDF Schema (RDFS) allows the expression of simple ontologies and the Web Ontology Language (OWL) can be used to represent knowledge about objects, groups of objects, and complex relations. Ontology languages allow users to write formal conceptualizations that must be syntactically and semantically well defined. They have high expressiveness and support for efficient inference. OWL is built on top of RDF and RDFS, with the same kind of syntax, whereas formal semantics and inference support are provided by mapping the ontology to a known logic formalism. OWL uses description logics and reasoners such as FACT, HERMIT and RACER [31]. SPARQL is a query language for RDF files based on pattern matching.

3. A Framework for Risk-Based Access Control in the Cloud

As discussed in Sections 1 and 2, there are standard definitions and authorization frameworks for traditional access control models, XACML being one of the most successful. On the other hand, there is a myriad of risk-based access control models employing different quantification and aggregation methods and no common enforcement approach capable of supporting several models.

In this Section, we present a framework based on the quantification and aggregation of risk metrics. Metrics are defined in risk policies, giving resource owners and cloud service providers greater control over the flexibility of authorization. Our framework is an extension of XACML and combines XACML policy evaluation with risk-based access control.

3.1. Definitions and Architecture

A risk metric ($r \in \mathbb{R}$) is a real value representing the risk associated to a certain characteristic of the system, user, resource, context or others. A risk policy $RP = \{\mathit{quantifyRisk}_1, \ldots, \mathit{quantifyRisk}_n, \mathit{aggregateRisk}, \mathit{riskThreshold}\}$ is a set containing a finite number n of quantification functions $\mathit{quantifyRisk}_i$, which take as input an access request (i.e. a subject s, an object o, an action a and a context c) and output a risk metric; one risk aggregation function *aggregateRisk*, which takes as input n real numbers and outputs another real number representing total risk; and the maximum risk accepted by the system *riskThreshold* $\in \mathbb{R}$. The access decision function in the framework is

$$canAccess(s, o, a, c) = \begin{cases} 1 & \text{if } combinePolicies(canAccessRisk(s, o, a, c), \\ & \qquad canAccessXACML(s, o, a)) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where *canAccessXACML* applies the usual XACML policy evaluation, and *canAccessRisk* is a modified version of (1) taking into account risk metrics:

$$canAccessRisk(s, o, a, c) = \begin{cases} 1 & \text{if } aggregatedRisk(s, o, a, c) < riskThreshold \\ 0 & \text{otherwise} \end{cases}$$

where *aggregatedRisk* is the application of the aggregation function to the risk metrics obtained by the quantification functions:

$$aggregatedRisk(s, o, a, c) = aggregateRisk(quantifyRisk_1(s, o, a, c), \ldots, \\ quantifyRisk_n(s, o, a, c))$$

The policy combining function *combinePolicies* takes the results of *canAccessRisk* and *canAccessXACML* and returns a single value. As before, a value of 1 indicates a granted request and a value of 0 a denied request. An example of a simple quantification function is a method that returns 0 for an HTTPS request and 1 otherwise. Some aggregation functions are minimum value, maximum value, average value, and the fuzzy method [32].

In practice, when the PDP receives an access request, it can perform two actions in parallel. On the one hand the XACML access control decision is taken based on the XACML policies related to the resource and on the other hand the PDP and a new component called Risk Engine perform a risk analysis of the access request, based on risk policies. Quantification functions can be either local, implemented directly in the Risk Engine, or remote, implemented by the user as a web service that is invoked during policy evaluation. Figure 1 shows an overview of the proposed framework. In the Figure, rounded rectangles (plus the circle and the cloud) represent the components of the system and arrows represent the communication between them. Components in purple belong to the XACML standard, whereas the following components (in green) are contributions of this paper. Details about the communication between components are described in Section 3.3.

Risk Engine: invoked by the PDP to process risk-based access control. It is responsible for analyzing and processing the risk policies associated to a resource and for invoking the quantification and aggregation functions described in each policy. The Risk Engine is different for each CSP because it implements locally the quantification functions available in that provider. If the user wants to use other functions, their implementation can be provided as a web service whose URL is informed in the risk policy;

Risk Quantification Functions: local functions that implement quantification for risk metrics. They are implemented inside the Risk Engine and are available to be used in the risk policies;

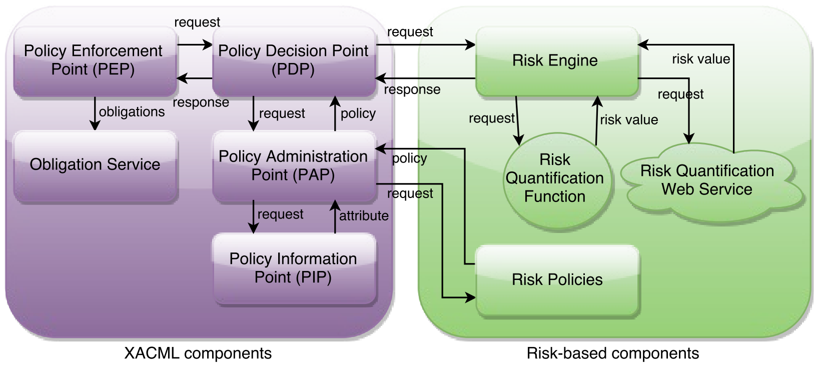

Figure 1: Overview of the architecture

Risk Quantification Web Services: web services responsible for quantifying the risk of each access request. They are implemented by users according to the specifications of the CSP. Each web service takes as input an access request forwarded by the risk engine and returns a numeric value that represents the quantification of the risk metric; and,

Risk Policies: policies that define how risk-based access control is evaluated for each resource.

The evaluation of risk policies has the same possible results as an XACML policy evaluation (Permit, Deny, NotApplicable and Indeterminate). After evaluating the policies, the PDP has two access decisions, one based on XACML and one based on risk. These decisions may be incompatible, so it is necessary to combine them in a final result using one of the following policy combining functions.

Deny Overrides: if any policy evaluates to Deny, the end result is Deny;
Permit Overrides: if any policy evaluates to Permit, the end result is Permit;
XACML Precedence: the end result is the same as the XACML policy; and,
Risk Precedence: the end result is the same as the risk policy.

3.2. Risk Policies

A risk policy is implemented as an XML file that describes to the CSP how to evaluate risk-based access control for a resource. This file is created by the resource owner and stored in the CSP. Each policy is composed of the identification of the associated resource, identification of the resource owner, a set of risk metrics with descriptions and quantification functions, a risk aggregation function and a risk threshold.

Both the CSP and the owner of the resource must opt in to have a resource accessed via a risk policy. Besides the risk policies defined by the owner, the CSP must also provide a basic risk policy that contains minimum metrics and a minimum risk threshold. The basic risk policy of each CSP is evaluated in every access request before the specific policies of each resource. If the basic policy is violated, the specific policies are not even processed. Basic policies are an important tool to keep the minimum security requirements of a CSP, at the same time allowing access control flexibility.

Risk policies support the use of different risk metrics and aggregation methods in the same system. Risk policies follow an XML schema with a root element *risk-policy* and the child elements *resource, user, metric-set, aggregation-function,* and *risk-threshold*. A *risk-policy* informs the version being used, while *resource* and *user* have *id* attributes identifying the associated resource and the policy creator, respectively. *aggregation-function* identifies the aggregation method being used and *risk-threshold* represents the maximum risk accepted by the system. Inside a *metric-set*, there are *metric* elements, i.e. risk metric definitions, and inside these the elements *name, description* and *quantification*, which can be a local function or a remote web service. Listing 1 shows a simple risk policy.

Listing 1: Example of a simple risk policy

```
<risk-policy version="1.0"
xmlns:rp="http://inf.ufsc.br/~danielrs/risk-policy">
<rp:resource id="0"/>
<rp:user id="0"/>
<rp:metric-set name="NAME">
  <rp:metric>
    <rp:name>NAME</rp:name>
    <rp:description>DESCRIPTION</rp:description>
    <rp:quantification>QUANTIFICATION</rp:quantification>
  </rp:metric>
</rp:metric-set>
<rp:aggregation-function>ABC</rp:aggregation-function>
<rp:risk-threshold>99</rp:risk-threshold>
</rp:risk-policy>
```

3.3. Decision Process

Figure 2 presents the access control decision process step by step. Again, purple elements represent XACML components, whereas green indicates components added by our framework. First, a subject issues a request to access a cloud resource (step 1), then the PEP receives this request and forwards it to the PDP (step 2), which loads the XACML and risk policies associated to the resource in the PAP (steps 3 and 4). At this point, the two access decisions happen in parallel. For XACML, the PDP loads the attributes from the PIP (steps 5 and 6) and performs the traditional access decision. For the risk decision, the PDP checks if the resource can be accessed this way, which is indicated by the presence of an associated risk policy. If such a policy does not exist, the result is NotApplicable. If it exists, the PDP forwards the request to the Risk Engine, which first checks the basic risk policy. If the basic policy evaluates to Permit, the Risk Engine performs the risk analysis described in the policy (steps 6 and 7). Steps 6c and 7c show a local risk quantification and steps 6b and 7b show the remote version (both versions can be invoked in the same policy). The risk metrics are then aggregated in a single value and the Risk Engine returns a response to the PDP. The PDP, based on the decisions of the XACML policy and the risk policy, as well as the combination algorithm, decides whether to

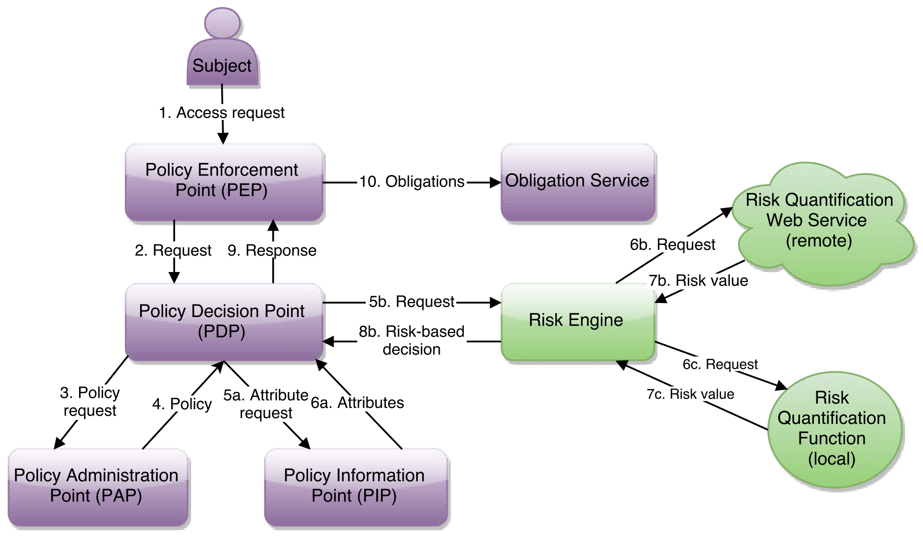

Figure 2: Decision process

grant or deny the request and returns a decision to the PEP (step 9), which is then responsible for enforcing obligations (step 10).

4. Instantiating the framework

An instantiation of the framework is characterized by defining the elements of a risk policy, namely metrics, quantification functions, an aggregation function and a risk threshold. In this Section, we instantiate the framework presented in the previous Section with three risk-based models: the one presented by Sharma et al. [33], the RAdAC model as described by Britton and Brown [25] and a custom model that combines characteristics from both. For each model we present the risk metrics, a few examples of quantification functions, and an aggregation function. Risk thresholds are system-specific, and therefore not shown, but the threshold should always be a value in the interval of possible values returned by the quantification function that represents how risk-averse is the system.

4.1. Confidentiality, Integrity, and Availability

The model of Sharma et al. [33] is based on the impact of a requested action on Confidentiality, Integrity, and Availability (CIA) and a past risk score added to the current value to keep track of user behavior. Table 1 presents the impact of actions on different kinds of data (where 1 means the action has an impact on the considered attribute and 0 otherwise). To instantiate this model, we need four risk metrics, Confidentiality (C), Integrity (I), Availability (A), and

History (H), and therefore four quantification functions, such as the following for the C, I, A metrics (based on Table 1):

$$quantifyRisk_C(s, o, a, c) = \begin{cases} 1 & \text{if } a = View \wedge o \in SENSITIVE \\ 0 & \text{otherwise} \end{cases}$$

$$quantifyRisk_I(s, o, a, c) = \begin{cases} 0 & \text{if } a = View \\ 1 & \text{otherwise} \end{cases}$$

$$quantifyRisk_A(s, o, a, c) = \begin{cases} 0 & \text{if } a = View \wedge o \in SENSITIVE \\ 1 & \text{otherwise} \end{cases}$$

The H metric is quantified by reading the previous risk value from a database and the final risk score is calculated based on the following formula:

$$aggregatedRisk = ((C \cdot P) + (I \cdot P) + (A \cdot P)) + H$$

where C, I, A is the impact of an action on confidentiality, integrity, and availability, respectively, as returned by the *quantifyRisk* functions above; H is the past risk value; and P is the probability of occurrence of the action (which should be known by the system).

An example of access request and decision using this model is when a user u, with a past risk score of 0.3, requests to *View* a *Sensitive* information ($C = 1, I = A = 0$, according to Table 1), in a system where the probability of this action is 0.5. The *aggregatedRisk* in this case is

$$aggregatedRisk = ((1 \cdot 0.5) + (0 \cdot 0.5) + (0 \cdot 0.5)) + 0.3 = 0.8 \qquad (2)$$

and access will be granted if the *riskThreshold* is greater than 0.8.

4.2. RAdAC

Britton and Brown [25] present a quantification method for the RAdAC model based on 27 risk factors divided in 6 groups, to which weights are attributed (shown in Table 2). We directly map each risk factor to a metric with the same name.

Group 1 (Characteristics of the Requester) represents risks associated with the person or application requesting access to a resource. Group 1 contains metrics such as the role of the requester in the system, its rank in the organization,

Table 1: Risk values from [33]

Action	**Data Sensitivity**	**C**	**I**	**A**
Create	Sensitive / Non-sensitive	0	1	1
View	Sensitive	1	0	0
View	Non-sensitive	0	0	1
Modify	Sensitive / Non-sensitive	0	1	1
Delete	Sensitive / Non-sensitive	0	1	1

its clearance level, its access level, possible previous violations, and education level. Group 2 (Characteristics of IT Components) is related to the risks associated with components in the path between requester and resource, such as the type of machines, applications, network, connections and authentication used. Group 3 (Situational Factors) are risks associated with the situation surrounding the request, such as the role of the requester in a specific mission and the time sensitivity of the requested resource. Group 4 (Environmental Factors) contains risks associated with the environment surrounding the request, such as the physical location of the requester. Group 5 (Characteristics of the Information Requested) are risks associated to the resource itself, such as classification level, permission level and encryption level. Group 6 (Heuristics) represents the risk associated to previous similar requests, such as known violations (risk knowledge) and successful transactions (trust level). Each group has a total weight of 16.6 (100/6), and each metric in a group has a weight of $16.6/n$, where n is the number of items in that group. Some metrics (e.g., threat level) are tailored for military applications, which is the original intended domain of the model. Details about all the metrics can be found in [25].

To instantiate this model, we use 27 quantification functions, such as the following, for the *Role* metric (the first metric in Table 2):

$$\mathit{quantifyRisk}_{\mathit{Role}}(s, o, a, c) = \begin{cases} 1 & \text{if } s \in \mathit{SuperAdmin} \\ 5 & \text{if } s \in \mathit{Admin} \\ 10 & \text{if } s \in \mathit{User} \\ 15 & \text{otherwise} \end{cases} \tag{3}$$

The final risk score is given by the formula:

$$\mathit{aggregatedRisk} = \sum_{i=1}^{27} w_i \cdot r_i \tag{4}$$

where w_i is the weight attributed to metric i and r_i is its risk value. The original work does not go into details on how to obtain the risk values for each factor. The values presented in (3) are just an example and different systems can adopt different values.

An illustrative example of access request and decision using this model is a request where all the *quantifyRisk* functions return a value of 5 for the metrics. The *aggregatedRisk* is

$$\begin{aligned} \mathit{aggregatedRisk} = (2.7 \cdot 6 \cdot 5) + (2.3 \cdot 7 \cdot 5) + (3.3 \cdot 5 \cdot 5) + \\ (8.3 \cdot 2 \cdot 5) + (3.3 \cdot 5 \cdot 5) + (8.3 \cdot 2 \cdot 5) = 492.5 \end{aligned} \tag{5}$$

where the terms in parentheses represent each group (e.g., group 1 has 6 metrics with weight 2.7 and the value of each is 5). Access will be granted if the *riskThreshold* is greater than 492.5.

4.3. Custom model

To further demonstrate the applicability of the framework and to combine interesting features from different models, we developed a custom model mixing

Table 2: Risk factors and associated weights— from [25]

Risk factor	Weight	Risk factor	Weight
Charact. of Requester	**16.6**	**Situational Factors**	**16.6**
Role	2.7	Specific Mission Role	3.3
Rank	2.7	Time Sensitivity of Inf.	3.3
Clearance Level	2.7	Transaction Type	3.3
Access Level	2.7	Auditable	3.3
Previous Violations	2.7	Audience Size	3.3
Education Level	2.7		
Charact. of IT Comp.	**16.6**	**Charact. of Inf. Req.**	**16.6**
Machine Type	2.3	Classification Level	3.3
Application	2.3	Encryption Level	3.3
Connection Type	2.3	Network Classif. Level	3.3
Authentication Type	2.3	Permission Level	3.3
Network	2.3	Perishable	3.3
QoP/Encryption Level	2.3		
Distance	2.3		
Heuristics	**16.6**	**Environmental Factors**	**16.6**
Risk Knowledge	8.3	Current Location	8.3
Trust Level	8.3	Op. Env. Threat Level	8.3

the 27 contextual metrics of [25], the 3 CIA metrics of [34] (which are the same as in [33]) and the historical metric of [33]. Therefore, we have 31 metrics, divided in three categories.

Context: metrics from [25], which include characteristics of the subjects, IT components, objects, environmental factors, situational factors, and heuristics;

Security characteristics of actions: confidentiality, integrity and availability impact of actions on the resource;

Subject history: a metric related to the history of actions of the user.

To estimate the risks involving confidentiality, integrity, and availability, we use the functions from [34], where the risk is based on the impact that an access can cause, which can be low (1-5), medium (6-10) or high (11-15). We use the same scale for the risk metrics that compose the contextual risk, such as the following function for the MachineType metric:

$$quantifyRisk_{MachineType}(s,o,a,c) = \begin{cases} 1 & \text{if } c[machineType] = Server \\ 5 & \text{if } c[machineType] = Desktop \\ 10 & \text{if } c[machineType] = Mobile \\ 15 & \text{otherwise} \end{cases}$$

The total risk is given by the formula:

$$aggregatedRisk = (w_1 \cdot contextRisk) + (w_2 \cdot ciaRisk) + (w_3 \cdot histRisk) \tag{6}$$

where *contextRisk* is obtained by applying (4), $ciaRisk = \sum_{x \in \{C,I,A\}} quantifyRisk_x$, *histRisk* is read from the database, and w_1, w_2, w_3 are weights attributed to each category of metrics.

Besides showing the expressiveness of our framework, this model is interesting because it combines metrics separately mentioned in several related works. It also illustrates the difficulty of having a single set of metrics that is applicable in several systems and how important it is to have an enforcement mechanism that accepts a varying set of metrics, so that system administrators can define their policies according to their own applications. It is easy to imagine several custom models that can be described simply by adding or removing a new category with its own set of metrics. It is also easy to see that these can be implemented in our framework by defining appropriate functions and policies.

An example of access request and decision using this model is the same request from formula (2), in the context given in formula (5), in a system where $w_1 = 0.2$, $w_2 = 0.7$, and $w_3 = 0.1$. The resulting risk is

$$aggregatedRisk = (0.2 \cdot 492.5) + (0.7 \cdot 0.5) + (0.1 \cdot 0.3) = 98.88$$

and access will be granted if the *riskThreshold* is greater than 98.88.

5. An Ontology-based Approach to Risk Calculation for the RAdAC Model

One of the biggest challenges in using RAdAC is finding a good risk estimation method. The method of [25] is a possibility, but the availability of contextual information to evaluate risk metrics is a major issue for risk quantification, since if some information is absent a final value cannot be achieved. Our ontology-based approach tries to solve this problem by adjusting the weights of each metric as they become available. Hence, at run-time, if there are few metrics available, they will have a greater weight and vice-versa. Furthermore, using the inference capabilities of ontology languages and reasoners, it tries to dynamically infer missing contextual information (in the form of attributes used in the metrics) that can be derived from available data.

5.1. Ontology definition

We use OWL-DL to construct ontologies. Relations between risk attributes can be defined using detailed knowledge about the architecture and behavior of a system and then used for automated inference. For instance, in a context composed of the metrics *ConnectionType, Role, MachineType,* and *Application*, taken from [25], the *Role* attribute can be obtained from the knowledge of other attributes. By knowing that the connection used is *Wired*, it is possible to look for machine types that use this type of connection and from the type of machine found it is possible to look for roles that use this type of machine. This example is formalized in the following formula:

$$ConnectionType(?d) \wedge MachineType(?m) \wedge canBeUsedBy(?d, ?m)$$
$$\wedge usesMachineType(?r, ?m) \rightarrow Role(?r)$$

Table 3: Relation properties

#	**RiskFactor:Attribute**	**RelationProperty**	**RiskFactor:Attribute**
1	ConnectionType:Wired	canBeUsedBy	MachineType:Desktop
2	Role:Admin	usesMachineType	MachineType:Desktop
3	MachineType:PDA	usesConnectionType	ConnectionType:Wireless
4	Role:TeamLeader	hasMinEducLevel	EducationLevel:Specialist
5	Role:TeamLeader	hasMinimumRank	Rank:E1
6	Application:Browser	usesEncryptionLevel	EncryptionLevel:SSL
7	TransactionType:Query	usesApplication	Application:Database
8	clearanceLevel:Secret	hasMinimumRole	Role:TeamLeader
9	RiskKnowledge:None	hasTrustLevel	TrustLevel:LowTrustLevel
10	CurrentLoc:Unknown	hasOpThreatLevel	OpThreatLevel:Severe
11	ClassifLevel:TopSecret	hasEncryptionLevel	EncryptionLevel:PKI

where d represents the type of connection being used, m represents the type of machine, and r represents the role of the requester. In this example, if the machine type is *Desktop* and the connection type is *Wired*, the role can be inferred as *Admin* (according to the example relations in Table 3).

In each row of Table 3, the first column represents a risk metric taken from Table 2 and associated to a possible attribute, such as connection type associated to wired (ConnectionType:Wired) in row 1. The third column represents another risk metric with another possible value, such as machine type associated to desktop (MachineType:Desktop) in row 1. The second column names a relation between an element in the first column and an element in the third column. The relation "canBeUsedBy" in row 1 formalizes that wired connections are used by desktop machines in the model. The relation "usesApplication" in row 7 formalizes that transactions of the type query use applications of type database. These relations are examples based on the metrics from [25] and an illustrative system. Other relations can be defined accordingly for different metrics and systems.

It is possible to write an OWL ontology mapping all risk metrics from a risk-based model to classes and sub-classes. In the case of [25], the result is an ontology with 27 classes and predefined weights associated to each class. To represent the risk weights as properties of the classes, the punning technique can be used, wherein objects are modeled both as classes and as instances, so that they can be validated according to the context in which they are used. In our example, all risk metrics are present also as instances, therefore classes and instances are denoted by the same URI. For instance, the URI `http://semanticweb.org/marinho/ontologies/2014/risk-ontology#machineType` represents both the *machineType* class and the *machineType* instance that has a *weight* data property. Another important modeling step is identifying synonyms of instances. For instance, the *SSL* attribute and the *Secure Socket Layer* attribute can be tied together by using the `owl:sameAs` property.

5.2. Access control evaluation

During access control evaluation, SPARQL queries are dynamically constructed from the available information and used to infer missing risk attributes in the context. An example is shown in Listing 2, where the attribute *Role* is derived from the available attributes *Desktop, HTTP,* and *PhD* and the relations *canBeUsedBy* and *usesMachineType* (defined in Table 3).

Listing 2: SPARQL query for attribute inference

```
SELECT ?riskFactor ?weight ?value
WHERE
  {risk:Desktop a ?riskFactor.
  ?riskFactor risk:weight ?weight.
  risk:Desktop risk:value ?value}
UNION
  {risk: Wired risk:canBeUsedBy ?MachineTypeRiskAtribute.
  ?riskAtribute risk:usesMachineType ?MachineTypeRiskAtribute.
  ?riskAtribute a ?riskFactor.
  ?riskFactor risk:weight ?weight.
  ?riskAtribute risk:value ?value}
UNION
  {risk:Wired a ?riskFactor.
  ?riskFactor risk:weight ?weight.
  risk:Wired risk:value ?value}
UNION
  {risk:Phd a ?riskFactor.
  ?riskFactor risk:weight ?weight.
  risk:Phd risk:value ?value}
```

The weights of each metric can be adjusted based on the available metrics according to the following formula:

$$W_a(x) = \left(\frac{\sum W_t}{\sum W_f} \right) \cdot W_f(x)$$

where $W_a(x)$ is the adjusted weight of metric x, $\sum W_t$ is the total weight of all the metrics, $\sum W_f$ is the total weight of the available metrics and $W_f(x)$ is the current weight of metric x. For instance, if only the risk metrics of the category *Characteristics of Requester* are available, the weight of each metric of the class (2.7777) is adjusted to:

$$\frac{100}{16.6666} \cdot 2.7777 = 16.666$$

Figure 3 describes the integration of the ontology-based approach with the framework presented in Section 3. The purple elements represent either XACML or framework-related components already presented, whereas the green elements are new components used in the ontology-based approach. The additions are the *Context Risk Ontology* file, storing an OWL ontology defined for the system and the *SPARQL Engine* used for querying the ontology. Some XACML components (e.g., PAP and PIP) are hidden to keep the Figure simple.

The decision process for this version of the framework is explained in the following (simplified because of the hidden XACML elements). A subject requests access to an application, which is intercepted by a PEP that sends to the PDP an XML message containing the context of the request (the Figure shows

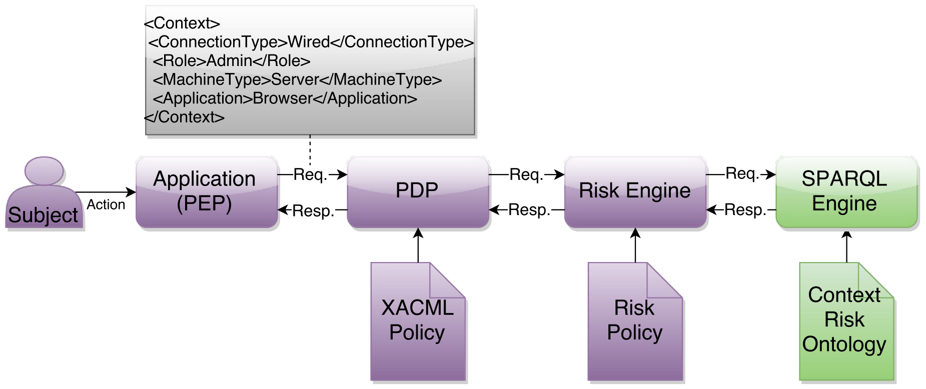

Figure 3: Overview of the ontology-based architecture

a simplified XML file, but the context is encapsulated in an XACML request). The context is composed of a series of attributes used in risk estimation and, in Figure 3, the example request has the attributes ConnectionType, Role, MachineType and Application. Processing of the XACML policy and the decision whether to use risk-based access control are omitted. When the access request reaches the Risk Engine, it processes the risk policy and uses the aggregation method described in (6). The Risk Engine is extended to dynamically build SPARQL queries as described above and, since it is possible that not all risk attributes are provided, automated inference is used to recover information and the weight of each metric is recalculated according to the available metrics, thus creating a dynamic context. The Risk Engine returns the aggregated risk to the PDP and the rest of the decision process is as described in Section 3.3.

6. Risk-based Access Control in Cloud Federations

A cloud federation aggregates services from different providers in a single set supporting three basic interoperability characteristics: resource migration, redundancy, and combination of complementary services or resources [35]. There are several proposals for cloud federation architectures, also called multi-clouds or clouds of clouds [36, 37]. Their idea is to present unique APIs, monitoring, and metering services, allowing organizations to join the federation [38].

One of the biggest challenges to create and manage a cloud federation is identity and access management [39]. To enforce access control, a CSP needs information about users that may come from another CSP and to establish trust in this scenario, the usual solution is to use an identity federation. An identity federation is a model of identity management where identity providers and service providers share user identities inside a circle of trust [40]. This solution presents problems such as the need of attribute and trust agreements, interoperability issues and, in practice, shows limited scalability [41].

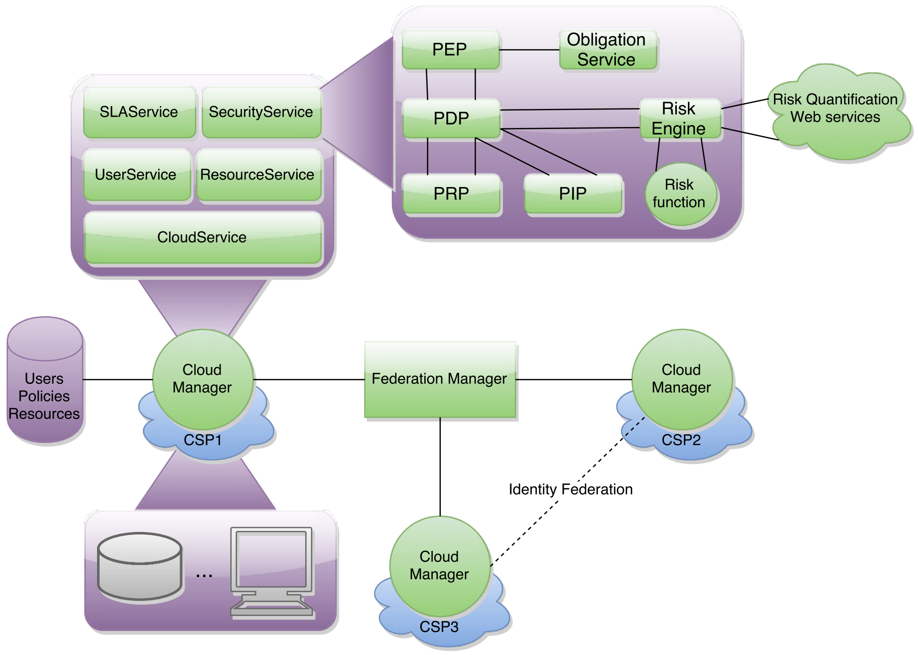

Figure 4: Overview of the cloud federation architecture

We focus on two of the aforementioned problems: (i) the need for trust agreements, and (ii) the need for attribute agreements. A risk-based access control model can be used to reduce both problems. For (i), it is simple to assign a risk metric to each attribute in a request that comes from a CSP that is not part of an identity federation, this can even be made dynamically with the ontology-based approach. As for (ii), it is possible to use the ontology-based approach presented in Section 5, to infer the missing attributes in a request and to define synonym relations between local attributes and foreign attributes. Thus, the use of the risk-based access control framework and instantiations presented in this paper allows access control in different CSPs without the use of an identity federation, fostering the formation of dynamic federations.

In this example scenario, we consider the cloud federation architecture shown in Figure 4. The architecture is simpler than a real world federation but focuses on basic components and shows the applicability of our access control proposal. The main components in this architecture are the *CSP*, which provides the infrastructure over which the resources are allocated; the *CloudManager*, which connects a CSP to the federation by providing the interoperability services; and the *FederationManager*, which groups the *CloudManagers* into a federation and manages message passing between members of the federation. The CloudManager is responsible for many interoperability services, among which the *SecurityService*, which contains the implementation of the framework presented in

Section 3. As shown in Figure 4 by a dashed line, some participating CSPs can form identity federations. We now describe two use cases of this architecture: resource instantiation and resource access.

Resource instantiation. When a user wants to instantiate a resource, e.g., a virtual machine or a virtual disk image, he/she can choose to do it on his/her own cloud or on another, foreign, cloud in the federation. When instantiating this resource, the user must also create an associated XACML access control policy and possibly a risk policy (by using an appropriate GUI in both cases). The risk policy represents the user's desire of having that resource accessed using risk-based access control. If both policies are created, the user must also define the combination algorithm to be used. To create the risk policy, a GUI containing a list of local quantification and aggregation functions should be presented to the user, as well as an interface where he/she can insert his/her own functions defined as web services.

Resource access. When a user tries to access a resource in his/her local cloud, the access control is performed as usual by the cloud management software (e.g., OpenStack). When a user tries to access a resource in a foreign cloud in the federation, the request is routed to the risk-based access control implementation in the SecurityService of the CloudManager of that CSP.

7. Implementation and Experiments

We developed two implementations of the framework, the first in Python, using the ndg-xacml[1] engine and the second in Java, using the HERAS-AF[2] engine. This choice was due to the better availability of ontology tools in Java and to faster development in Python. The engines were chosen because they are both open source, with tests and documentation available.

In the Python implementation, we used the web.py framework for the web services, whereas in the Java implementation we used JSP 2.1 for an example application, and the SPARQL library available in Apache Jena[3]. All the experiments report the minimum, maximum and mean average (out of 10 runs) time (in milliseconds) taken to reach an access decision using different policies in the framework.

Appendix A shows the implementation of the risk policy, one risk quantification function and the risk aggregation function for the model described in Section 4.1. Policies for other models are similar but larger. Other quantification functions are similar, taking as input contextual information and returning a value representing its risk.

[1]`https://pypi.python.org/pypi/ndg-xacml/0.5.1`
[2] `http://www.herasaf.org/`
[3]`https://jena.apache.org/`

7.1. Performance of the instantiations

The instantiations described in Sections 4.1 and 4.2 were tested in the Python implementation, whereas the custom model and the ontology-based approach were tested in the Java implementation. Table 4 shows the time spent to reach an access decision using three different policies: only XACML (0 risk metrics), XACML+[33] (4 risk metrics), and XACML+[25] (27 risk metrics). In this experiment, all the quantification and aggregation methods are local. As expected, due to the number of metrics, XACML only is the fastest and XACML+[25] is the slowest. Table 5 shows the results for the custom model (31 risk metrics). In this experiment, we measured the time taken to reach an access decision (Decision row in the Table), the time taken for the example application to receive and forward the access request (App row), and the time taken from the moment the user requests the page in the browser until it is fully loaded (Browser row).

To test the ontology-based approach, we used Protégé[4] to define the OWL ontology representing the custom model and the relations shown in Table 3. Several instances of the ontology classes were created to allow for the inference of missing metrics. First, to validate the approach, we developed a base scenario using our custom model enriched with the ontology. We then measured the risk in four distinct cases. First, with the 31 metrics (R), then with 7 missing metrics (R_m), then with the same missing metrics but adjusting the weights (R_a) and then using metric inference and adjusting the weights (R_i). The results are shown in Figures 5 (only considering contextual risk) and 6 (considering the total risk), where the y axis stands for the risk value and the x axis shows the four cases described above (R in blue, R_m in red, R_a in green and R_i in purple). It is noticeable that the use of inference and weight redistribution brings the estimated risk closer to the base risk than when none of the approaches is used. The minimum time spent querying the ontology was 150 ms, the maximum time was 200 ms and the average time was 178.9 ms. The time taken to modify a policy and the reasoning time were not measured because they are done offline, since the policy is only modified by a system administrator and not for every access request. The time that the reasoner takes to parse the ontology structure is also out of the scope of the access requests.

[4]`http://protege.stanford.edu/`

Table 4: Performance of risk policies

Policy	**min**	**max**	**avg**
XACML	0.925	4.278	1.040
XACML+[33]	1.986	11.973	2.436
XACML+[25]	4.395	14.234	5.352

Table 5: Performance of the custom model

Measure	**min**	**max**	**avg**
Browser	970	1060	1570
App	840	897	1129
Decision	442	465	524

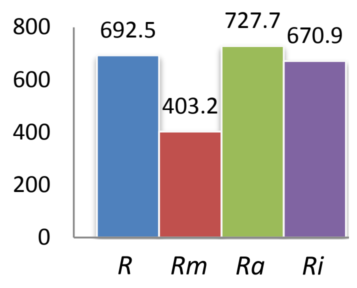

Figure 5: Ontology experiments (contextual risk)

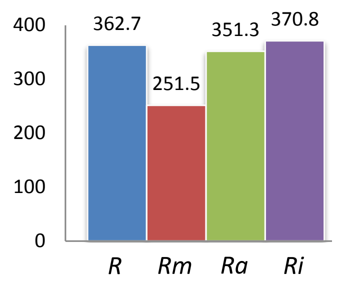

Figure 6: Ontology experiments (total risk)

7.2. Number of metrics

We also explored the impact of the number of metrics in a risk policy. In the first experiment, there is a varying number of metrics with local quantification. To focus on the number of metrics and not their complexity, all the quantification functions used were the same, simply returning a random number for the risk value. The results are shown in Table 6. It is clear that increasing the number of metrics degrades performance, but this degradation is smooth and even with many metrics (10000) the average time is around 1.5 seconds, which is reasonable. It is also important to highlight that a great part of this time is due to processing the XML policy and not to the quantification functions themselves.

Next we study the use of remote quantification with web services. In this experiment, we used a risk policy with 10 metrics (with more than 10 remote metrics, performance becomes totally unacceptable). To test the impact of remote metrics in the same policy, we defined 4 test cases. The first case (A) contains only XACML with no risk metrics, for reference. The second (B) contains 10 local quantification metrics. The third (C) contains 5 local and 5 remote metrics and the fourth (D) contains 10 remote metrics. Table 7 shows the results of all test cases. It is clear that the use of web services severely affects performance and that the use of only 10 remote functions (test case D) is already inadequate (with an access decision time of 4.2 seconds on average). Figure 7 shows the time taken for an access decision in milliseconds (y axis) growing with the number of metrics (x axis). The solid line with square markers shows

Table 6: Performance with a different number of metrics (ms)

Metrics	**min**	**max**	**avg**
1	1.832	12.130	2.243
10	2.612	12.876	3.171
100	10.922	60.442	14.030
1000	96.041	175.245	121.383
10000	1168.511	1517.364	1361.025

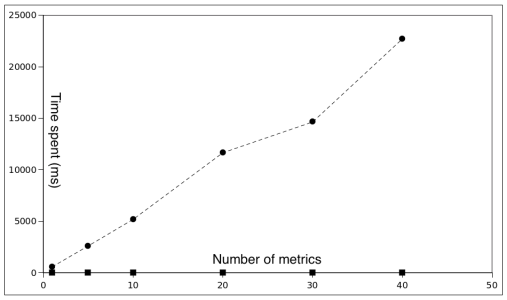

Figure 7: Performance of local and remote functions

this growth for local metrics and the dashed line with round markers shows the growth for remote functions.

7.3. Discussion

Measuring security. There is no standard quantitative measure for the security of an access control system. Such a system can be proven to be safe by defining its set of possible states and showing that there is no state where permissions can leak to unauthorized subjects, which is undecidable [42]. Hu et al. [42] recommend qualitative metrics to assess access control systems. The metrics are based on administrative capabilities and costs, policy coverage, extensibility, and performance. The administrative capabilities and costs of our framework are the same as XACML, with the added cost to manage risk policies. The coverage of policies and extensibility of the model were shown in the experiments by implementing models described in related work [25, 33]. Performance was tested in the experiments and reported in Tables 4 to 7 and in Figures 5 to 7.

Number of metrics. The experiments in Section 7.2 clearly report a loss of performance as the number of metrics used in a system grows. However, a larger number of metrics also allows a policy administrator to have more fine-grained control over the risk components involved in an access request. In the end, it is not possible to recommend a generally acceptable number of metrics, just as it is not possible to recommend a general number of roles and permissions in RBAC or a general number of attributes in ABAC. These numbers depend on the system being used, the desired level of risk granularity and the desired performance.

Table 7: Performance with remote quantification (ms)

Test case	**Metrics**	**min**	**max**	**avg**
A	XACML only	1.057	9.372	1.46
B	10 local	1.824	15.564	4.574
C	5 local + 5 remote	1556.182	2813.56	1726.71
D	10 remote	3247.563	10350.5	4220.6

8. Related work

Risk-based Access Control. Fall et al. [15] discussed the inadequacy of current access control models for multi-tenant clouds and proposed the use of RAdAC. Although their work introduced risk-based access control for the cloud, there was no validation of the idea. Arias-Cabarcos et al. [32] described challenges for federated identity management in the cloud, especially trust agreements, and proposed a risk evaluation methodology to enable dynamic identity federations. The authors proposed a set of metrics and a fuzzy aggregation function, but the work lacks a reference for the values of the metrics. The metrics are prefixed in the system and users cannot provide their own quantification or aggregation methods. Chen et al. [43] proposed the encoding of risk mitigation policies based on the XACML RBAC profile to incorporate risk management into an RBAC solution with support for obligations. The paper does not define risk quantification functions and there is no way to combine or aggregate the evaluation results of risk policies with the XACML results. Gasparini [16] extended the previous work introducing a risk-aware group-based access control model and applying the extension of XACML [43] in a health care scenario. Choi et al. [44] presented another extension of XACML to support risk-based access control, however their work is limited to medical information systems.

Ontologies for access control. Dersing et al. [8] developed a model that uses semantic contexts and ontologies to dynamically determine an appropriate role assignment for an incident handling system. Finin et al. [7] studied the relation between OWL and RBAC, showing two ways to support RBAC in OWL and discussing how they can be extended to attribute-based models. Tsai and Shao [45] used ontologies to build a hierarchy of roles for a specific RBAC domain. Bernabe et al. [46] described an authorization model encompassing several RBAC characteristics that uses semantic web for the access rules.

Cloud federations. In [47, 48, 49] a federation architecture was proposed, using a component integrated in the CSPs. Based on this architecture, Celesti et al. [50] proposed a federated identity management mechanism using a third-party identity provider. Contrail is a framework for building cloud federations with authentication and authorization, supporting, e.g., UCON [51].

This work. Our proposal allows the definition of flexible risk policies independent of the type of access control model used, the use of different risk metrics, the aggregation of various risk quantification functions to achieve a final risk value to be compared with a risk threshold, and the application of the framework in dynamic cloud environments and cloud federations. An ontology-based approach helps in calculating risk according to the context and in adjusting the weights of each metric considering the actual number of metrics, which is another novel contribution.

9. Discussion and Conclusions

We proposed a framework for risk-based access control, based on an extension of XACML and the use of risk policies, which adds flexibility for resource sharing in a dynamic and collaborative environment such as the cloud. The framework has as main advantages the possibility of using risk metrics, quantification and aggregation functions from different sources, including those defined by the user, and the use of basic risk policies to maintain minimum security requirements. Another important feature is the possibility of having distributed risk engines and enforcement points, as in XACML. We also described the implementation of two prototypes that showed satisfactory experimental results.

The expressiveness of our proposal was demonstrated by its instantiation with two risk-based models found in the literature, the development of a custom model and the proposed use of ontologies as a tool to enhance these models. We also identified a possible usage scenario, namely the integration of the framework in a cloud federation platform to allow the establishment of cloud federations without the use of identity federations.

The main limitations of the proposal are the overhead from processing risk policies and especially the performance degradation when using remote quantification functions. Performance improvements could be obtained by using JSON for representing policies [52], decision diagrams for XACML policy evaluation [53], parallel processing of remote functions and caching of access decisions.

As future work, we intend to integrate the access control model in a mature cloud federation project; implement other risk quantification methods to evaluate the need for new components; and develop a reference set of risk metrics for the cloud.

References

[1] P. Mell, T. Grance, SP 800-145. The NIST Definition of Cloud Computing, Tech. rep., NIST, Gaithersburg, MD, United States (2011).

[2] K. Ren, C. Wang, Q. Wang, Security Challenges for the Public Cloud, IEEE Internet Computing 16 (1) (2012) 69–73.

[3] OASIS, A Brief Introduction to XACML, Available at: `https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html`. Last accessed: 5th July, 2016 (2003).

[4] R. McGraw, Risk-Adaptable Access Control (RAdAC), in: NIST Privilege (Access) Management Workshop, 2009.

[5] D. dos Santos, C. Westphall, C. Westphall, Risk-based Dynamic Access Control for a Highly Scalable Cloud Federation, in: Proc. SECURWARE, 2013, pp. 8–13.

[6] A. Karp, H. Haury, M. Davis, From ABAC to ZBAC: The Evolution of Access Control Models, Tech. rep., IBM (2009).

[7] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, B. Thuraisingham, ROWLBAC: Representing Role Based Access Control in OWL, in: Proc. SACMAT, 2008, pp. 73–82.

[8] A. Dersingh, R. Liscano, A. Jost, J. Finnson, Dynamic Role Assignment Using Semantic Contexts, in: Proc. WAINA, 2009, pp. 1049–1054.

[9] D. R. dos Santos, C. Westphall, C. Westphall, A Dynamic Risk-based Access Control Architecture for Cloud Computing, in: Proc. NOMS, 2014, pp. 1–9.

[10] R. Marinho, C. Westphall, G. Schmitt, A Dynamic Approach to Risk Calculation for the RAdAC Model, in: Proc. SAM, 2014, pp. 83–88.

[11] M. Benantar, Access Control Systems: Security, Identity Management and Trust Models, Springer, 2006.

[12] V. Hu, D. Kuhn, D. Ferraiolo, Attribute-based Access Control, Computer 48 (2) (2015) 85–88.

[13] G. Zhang, M. Parashar, Dynamic Context-aware Access Control for Grid Applications, in: Proc. Grid, 2003, pp. 101–108.

[14] G. Zhang, M. Parashar, Context-aware dynamic access control for pervasive applications, in: Proc. CNDS, 2004, pp. 21–30.

[15] D. Fall, G. Blanc, T. Okuda, Y. Kadobayashi, S. Yamaguchi, Toward Quantified Risk-Adaptive Access Control for Multi-tenant Cloud Computing, in: Proc. JWIS, 2011.

[16] L. Gasparini, Risk-Aware Access Control and XACML, Ph.D. thesis, University of Padova (2013).

[17] Y. Li, H. Sun, Z. Chen, J. Ren, H. Luo, Using Trust and Risk in Access Control for Grid Environment, in: Proc. SECTECH, 2008, pp. 13–16.

[18] N. Diep, S. Lee, Y. Lee, H. Lee, Contextual Risk-Based Access Control, in: Proc. SAM, 2007, pp. 406–412.

[19] A. Ahmed, N. Zhang, An Access Control Architecture for Context-Risk-Aware Access Control: Architectural Design and Performance Evaluation, in: Proc. SECURWARE, 2010, pp. 251–260.

[20] N. Dimmock, How Much is "enough"? Risk in Trust-based Access Control, in: Proc. WETICE, 2003, pp. 281–282.

[21] A. Brucker, H. Petritsch, Extending Access Control Models with Break-glass, in: Proc. SACMAT, 2009, pp. 197–206.

[22] V. Suhendra, A survey on access control deployment, in: Proc. FGIT-SecTech, 2011, pp. 11–20.

[23] R. Shaikh, K. Adi, L. Logrippo, Dynamic Risk-based Decision Methods for Access Control Systems, Computers & Security 31 (4) (2012) 447–464.

[24] JASON Program Office, Horizontal Integration: Broader Access Models for Realizing Information Dominance, Tech. rep., MITRE Corporation (2004).

[25] D. Britton, I. Brown, A Security Risk Measurement for the RAdAC Model, Master's thesis, Naval Postgraduate School (2007).

[26] P. Cheng, P. Rohatgi, C. Keser, P. Karger, G. Wagner, A. Reninger, Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control, in: Proc. IEEE S&P, 2007, pp. 222–230.

[27] Q. Ni, E. Bertino, J. Lobo, Risk-based access control systems built on fuzzy inferences, in: Proc. ASIACCS, 2010, pp. 250–260.

[28] I. Molloy, L. Dickens, C. Morisset, P. Cheng, J. Lobo, A. Russo, Risk-based Security Decisions Under Uncertainty, in: Proc. CODASPY, 2012, pp. 157–168.

[29] Q. Wang, H. Jin, Quantified Risk-adaptive Access Control for Patient Privacy Protection in Health Information Systems, in: Proc. ASIACCS, 2011, pp. 406–410.

[30] W. Borst, Construction of Engineering Ontologies for Knowledge Sharing and Reuse, Ph.D. thesis, University of Twente (1997).

[31] S. Staab, R. Studer, Handbook on Ontologies, Springer, 2009.

[32] P. Arias-Cabarcos, F. Almenaarez-Mendoza, A. Maron-Lopez, D. Diaz-Sanchez, R. Sanchez-Guerrero, A Metric-Based Approach to Assess Risk for "On Cloud" Federated Identity Management, J. of Net. and Sys. Man. 20 (2012) 513–533.

[33] M. Sharma, Y. Bai, S. Chung, L. Dai, Using Risk in Access Control for Cloud-Assisted eHealth, in: Proc. HPCC, 2012, pp. 1047–1052.

[34] P. Saripalli, B. Walters, QUIRC: A Quantitative Impact and Risk Assessment Framework for Cloud Security, in: Proc. CLOUD, 2010, pp. 280–288.

[35] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, M. Kunze, Cloud Federation, in: Proc. CLOUD COMPUTING, 2011, pp. 32–38.

[36] M. AlZain, E. Pardede, B. Soh, J. Thom, Cloud Computing Security: From Single to Multi-clouds, in: Proc. HICSS, 2012, pp. 5490–5499.

[37] M. Vukolić, The Byzantine Empire in the Intercloud, SIGACT News 41 (3) (2010) 105–111.

[38] P. Harsh, Y. Jegou, R. Cascella, C. Morin, Contrail virtual execution platform challenges in being part of a cloud federation, in: Proc. ServiceWave, 2011, pp. 50–61.

[39] D. Sriram, Federated Identitiy Management in Intercloud, Master's thesis, TU Munchen (2013).

[40] H. Lee, I. Jeun, H. Jung, Criteria for Evaluating the Privacy Protection Level of Identity Management Services, in: Proc. SECURWARE, 2009, pp. 155–160.

[41] K. Lampropoulos, S. Denazis, Identity Management Directions in Future Internet, IEEE Communications Magazine 49 (12) (2012) 74–83.

[42] V. Hu, D. Ferraiolo, D. Kuhn, Assessment of Access Control Systems, Interagency Report 7316, Tech. rep., National Institute of Standards and Technology (2006).

[43] L. Chen, L. Gasparini, T. J. Norman, XACML and Risk-Aware Access Control, in: Proc. ICEIS, 2013, pp. 66–75.

[44] D. Choi, D. Kim, S. Park, A framework for context sensitive risk-based access control in medical information systems, Comp. Math. Methods in Medicine 2015 (2015) 265132:1–265132:9.

[45] W. Tsai, Q. Ssao, Role-Based Access-Control Using Reference Ontology in Clouds, in: Proc. ISADS, 2011, pp. 121–128.

[46] J. Bernabe, J. Perez, J. Calero, F. Clemente, G. Perez, A. Skarmeta, Towards an authorization system for cloud infrastructure providers, in: Proc. SECRYPT, 2011, pp. 333–338.

[47] A. Celesti, F. Tusa, M. Villari, A. Puliafito, Security and Cloud Computing: InterCloud Identity Management Infrastructure, in: Proc. WETICE 2010, 2010, pp. 263–265.

[48] A. Celesti, F. Tusa, M. Villari, A. Puliafito, Three-Phase Cross-Cloud Federation Model: The Cloud SSO Authentication, in: Proc. AFIN, 2010, pp. 94–101.

[49] A. Celesti, F. Tusa, M. Villari, A. Puliafito, Federation Establishment Between CLEVER Clouds Through a SAML SSO Authentication Profile, Int. J. on Adv. in Internet Tech. 4 (1).

[50] A. Celesti, F. Tusa, M. Villari, A. Puliafito, How to Enhance Cloud Architectures to Enable Cross-Federation, in: Proc. CLOUD, 2010, pp. 337–345.

[51] M. Coppola, P. Dazzi, A. Lazouski, F. Martinelli, P. Mori, J. Jensen, I. Johnson, P. Kershaw, The Contrail Approach to Cloud Federations, in: Proc. ISGC, 2012.

[52] OASIS, JSON Profile of XACML 3.0 Version 1.0, Available at: `http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cs01/xacml-json-http-v1.0-cs01.html`. Last accessed: 5th July, 2016 (2014).

[53] C. Ngo, Y. Demchenko, C. de Laat, Decision Diagrams for XACML Policy Evaluation and Management, Computers & Security 49 (0) (2015) 1–16.

Appendix A. Example of risk policy and metrics used in the experiments

Listing 3: Risk policy for the model described in 4.1

```
<rp:risk-policy version="1.0" xmlns:rp="http://inf.ufsc.br/~danielrs">
  <rp:resource id="1"/> <rp:user id="1"/>
  <rp:metric-set name="sharma2012">
    <rp:metric>
      <rp:name>Confidentiality</rp:name>
      <rp:description>Confidentiality cost</rp:description>
      <rp:quantification>https://localhost:8443/QuantifyConfidentiality
      </rp:quantification>
    </rp:metric>
    <rp:metric>
       <rp:name>Availability</rp:name>
       <rp:description>Availability cost</rp:description>
       <rp:quantification>https://localhost:8443/QuantifyAvailability
       </rp:quantification>
    </rp:metric>
    <rp:metric>
       <rp:name>Integrity</rp:name>
       <rp:description>Integrity cost</rp:description>
       <rp:quantification>https://localhost:8443/QuantifyIntegrity
       </rp:quantification>
       </rp:metric>
    </rp:metric-set>
    <rp:aggregation-function>https://localhost:8443/Aggregate
    </rp:aggregation-function>
    <rp:risk-threshold>1.5</rp:risk-threshold>
</rp:risk-policy>
```

Listing 4: Risk quantification and aggregation for the model described in 4.1

```
class QuantifyAvailability:
    def GET(self):
        params = web.input(user=None, action=None, resource=None)
        action = params.action
        resource = params.resource

        impact = 1
        if action == "VIEW" and resource not in sensitive:
            impact = 0

        web.setcookie("availability", impact)

#... (other metrics are not shown)

class Aggregate:
    def GET(self):
        a = float(web.cookies().get("availability"))
        i = float(web.cookies().get("integrity"))
        c = float(web.cookies().get("confidentiality"))
        h, p = (r.random(), r.random())

        return ((a*p) + (i*p) + (c*p) + h)
```