A Risk Calculus Extension to the XACML Language

Jhonatan Alves
Federal University of Santa Catarina
Networks and Management Laboratory
Florianópolis, Brazil
jhonatan.alves@posgrad.ufsc.br

Carla Merkle Westphall
Federal University of Santa Catarina
Networks and Management Laboratory
Florianópolis, Brazil
carlamw@inf.ufsc.br

Gustavo Roecker Schmitt
Federal University of Santa Catarina
Networks and Management Laboratory
Florianópolis, Brazil
gustavorschmitt@grad.ufsc.br

ABSTRACT

The increase of dynamic cloud computing environments introduces the need for new ways of access control in applications. One access control model which adapts flexibly to such systems on the Internet is the RAdAC (*Risk-Adaptive Access Control*). This model is based on the user confidence degree and the risk of releasing access to some information taking into account the context in which a request is performed. However, in practice, to use such model it is necessary to implement a technological support as, for example, extending the access control architecture present in the XACML (*eXtensible Access Control Markup Language*). This paper extends the XACML access control architecture to support the RAdAC model providing a quantitative, concrete and dynamic risk calculus in order to improve the access control in cloud environments. A prototype was developed in Amazon EC2 cloud environment to perform dynamic access control policies using the proposed XACML extension. Some risk calculus tests are described in the paper to exemplify the RAdAC decisions.

Categories and Subject Descriptors

K.6 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: [Security and Protection,Authentication,Unauthorized access]

General Terms

Security, Reliability, Availability, Confidentiality

Keywords

XACML, RAdAC, Risk Calculus, Cloud Computing

1. INTRODUCTION

Cloud computing is a new computational paradigm which provides IT resources as services. This paradigm improves the virtualization efficiency of data centers making IT resources more availed [6]. According to NIST (*National Institute of Standards and Technology*) [11] a cloud computing model should have a set of essential characteristics which includes on-demand self-service, broad network access, dynamic resource provisioning, measured service and elasticity of resources. Moreover, cloud computing brings lots of benefits to its users since it eliminates the need for installing, configuring, managing and updating resources. However, there are a wide number of challenges related to the security of cloud environments as, for example, data protection, malicious behavior of insiders, service unavailability, handling of security incidents, among others [1].

Since resources provided by cloud environments can be accessed by different entities, then appropriate and dynamic access control mechanisms are required to provide an improved security degree to such resources. A dynamic access control model which adapts flexibly to cloud computing environments is the RAdAC model. It uses risk calculus at the request time to allow or deny access request, since assessing the risk value is an essential tool to protect cloud environments from negative events.

However, it is necessary to implement the RAdAC model in order to use its access control facilities. One way of doing it is through extending the XACML language. This language allows to define a set of policies which delimits the actions an individual can perform describing the general requirements for access control. It also allows to specify requests and responses to aid the decision-making during authorization process. The RAdAC model (*Risk-Adaptive Access Control*) uses risk calculus at the request time to allow or deny the access request.

In this paper, we present an XACML extension to provide support to the RAdAC model, enabling a quantitative, concrete and dynamic risk calculus to be used in cloud computing environments. In the literature, there is a lack of risk calculus proposals to access control in the cloud, which is a propitious scenario to implement a real RAdAC application. Besides, some works [8] [4] consider the *dynamic* aspect of RAdAC as the possibility of using a set of factors, which compose the context, at the request *time*, to decide whether the access is allowed or denied. However, a particular context can repeat over time, thus the answer to a request in such context will be always the same independently of advancing time. Thus, there is a need for a new factor capable of evaluating contexts which repeat over time differentiating the access permissions as the time advances.

To solve this issue, we introduce into the risk calculus the notion of *user rank* which expresses numerically the user behavior over time. As the time advances, the user rank can change and influences directly the risk calculus outcome. If in a given context at time t an user does not have access to a resource, he may have access to such resource in the same context at time t', with $t < t'$, if the value of his user rank increases enough to make it possible.

This paper is organized as follows. Section 2 describes related

SBSI 2016, May 17th-20th, 2016, Florianópolis, Santa Catarina, Brazil

works, the XACML language and its architecture are presented in section 3, section 4 presents proposals for risk calculus, section 5 presents the proposal of this paper, section 6 describes some tests and section 7 presents the final considerations.

2. RELATED WORKS

The XACML language was designed insomuch that its extension could be as flexible as possible, allowing the introduction of new components in its data flow architecture and the addition of new elements in its XML Schema. This section presents a set of works that have extended the XACML language.

Chen Chen et al. (2010) proposed a set of modules to continuously control the risk calculus. If the access is allowed, then the risk is recalculated from time to time, considering changes in the context throughout the session. The access is canceled whether the risk's temporal distribution exceeds a certain threshold. Although the idea is quite interesting, the authors do not present a concrete formula of how risk should be calculated, they just claim it is a measurement at a time t which combines the cost associated to the resource to be consumed, the damage probability of the risk and its temporal distribution that is calculated in previous moments.

Liang Chen et. al (2013) extended the XACML language to support the RAAC model (*Risk-Aware Access Control*). Since the variables associated to entities involved in the request can be expressed as attributes, the risk associated with such attributes is rescued by PIP block (responsible for storing and controlling any attributes of individuals, resources, actions and environment), becoming part of the risk calculus in such context. A risk mitigation strategy is used to specify which obligations must be executed if a request in a given context has an acceptable risk, this is, if such risk is in $[k,k')$. However, the authors do not present how exactly the risk is calculated.

Daniel Ricardo dos Santos et. al (2014) extended the XACML architecture adding a set of new blocks, where *Risk engine* is responsible for analyzing and processing the policies associated with a resource, *Web Service* is responsible for quantifying the access risk when the user defines the method for calculating it and *Risk Policies* defines how the risk-based access control should be assessed for each resource. When an access request is received, the decision point can run in parallel two decisions: a decision based only on user attributes following XACML related policies and/or a decision based on risk following the risk policies. However, the authors do not present a concrete formula for risk calculus, leaving the task of defining risk policies to the administrator or user.

The literature reports few studies extending XACML language with support for risk calculus. Most of them is concerned to present new components, how they work, test scenarios and factors that should be considered during the calculus, but they do not show a *concrete way* of how risk must be calculated.

3. THE XACML LANGUAGE

tha XACML language is a XML-based language which allows to create flexible policies to describe the requirements for accessing resources. Among the various proposals for languages and access authorization architectures, XACML stands out as an international standard approved by OASIS, being a generic and extensible model, having a distributed architecture and support for multiple data types and functions [12]. The figure 1 illustrates the XACML architecture.

The execution flow begins at PEP block (*Policy Enforcement Point*) which is responsible for intercepting all access requests and forward them to the *Context Handler* block. This block converts the request into a *standard XACML request* which describes its *context*.

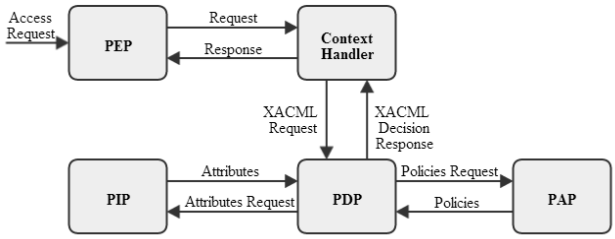

Figure 1: XACML data flow architecture.

The context of a request is composed by the *subject* who performs the request, the *resource* he wants to access, the *action* to be executed and the *environment* from where the request comes. This request is sent to the PDP block (*Policy Decision Point*) which decides whether *allows* or *denies* the access request. To this end, it asks PAP block (*Policy Administration Point*) the applicable set of policies to the request and asks PIP block (*Policy Information Point*) any additional attributes about the subject, resource, action or environment.

The final answer, the XACML decision response, is composed by the *decision* (allow or deny), *state* (determines if there was an error during the authorization process) and *obligations* (set of operations that must be performed by the PEP block when the request is allowed). The answer is forwarded to *context handler* which converts it into the PEP format and then sends the answer to PEP block.

4. RISK CALCULUS

Risk may be defined as $R = P\ x\ I$, where P determines the probability of the occurrence of an event and I represent the impact of this event. The risk calculus allows to determine whether an event can cause some damage to a system and its estimation is useful to take decision in order to avoid such damages. Various approaches have been proposed to determine effective mathematical means for estimating risk.

A model for using risk in access control tasks is the RAdAC *(Risk-Adaptive Access Control)* [13]. The execution data flow of this model is expressed in figure 2. In RAdAC model, the risk of an access request is calculated and then compared with a threshold of applicable policies over such request. Then, the operational need is determined, when specified by the applicable policies. The operational need is represented by a value and is a special requirement of an user to access certain information in order to complete a mission or execute a task. Usually, a nurse would not have access to all medical information of a patient, but, when a patient is having a heart attack, for example, a nurse is in a critical moment and should have access to all information necessary to save the patient's life. This represents a situation of operational need, when there is a new access control rule because of the context and moment. Thus, the access is allowed if the operational need is bigger than the risk value, otherwise it is denied [7]. In cases where the risk is not acceptable but the operational need is, then the access is allowed, if the applicable policies allow the operational need to override the risk.

The risk calculus is not a simple task and it is still a research subject. In [8] some practical challenges to achieve the implementation of RAdAC are described, such as the real-time calculus of security risk for each access decision, determination of operational risk, quantification of confidence level, use of heuristics to achieve access decisions and the possibility of revocation of access at any time. However, in relation to the risk calculus the paper just men-

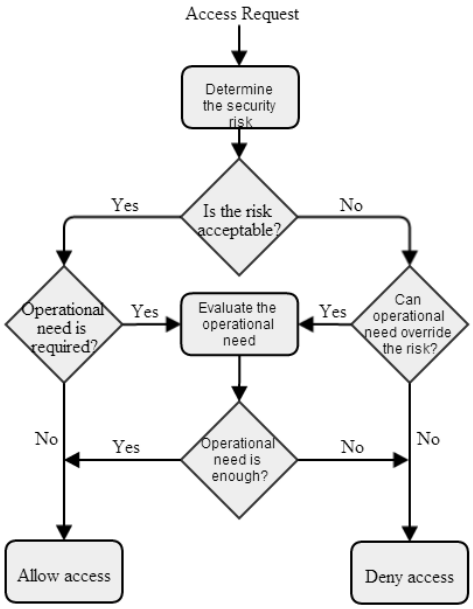

Figure 2: RAdAC data flow (adapted from [7]).

tions in which moments the risk must be calculated leaving open how to calculate it.

A formal model for calculating risk associated with the action will be taken over resources, considering features as *confidentiality*, *integrity* and *availability* is described in [4]. The authors propose a framework which uses context parameters, from where the access is coming, to calculate the risk. Figure 3 illustrates such framework.

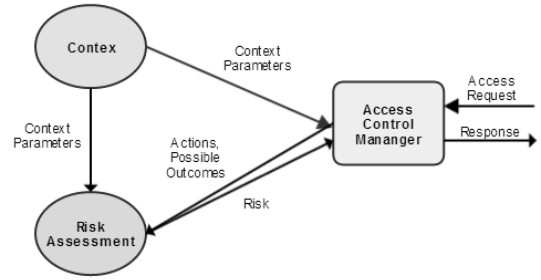

Figure 3: Framework architecture proposed in [4].

The access requests are received and analyzed by the main module, *Access control manager*, which sends them to *Risk Assessment* module along with context parameters collected by *Context* module. In *Risk Assessment*, the risk calculus is performed and then forwarded to the main module which will decide whether accepts or denies the access request.

We adapted and implemented the formalisms of block *Risk Assessment* in our architecture to determine the risk calculus at the access request time. Such formalisms are presented in section 5.2.3.

5. THE XACML EXTENSION

Our XACML extension proposal, described in this section, was developed by using Heras implementation [10]. The XACML standard requests and its architecture were modified to provide RAdAC support. The *Risk Assessment* module was defined with some mathematic formalism to perform the risk calculus.

5.1 The XACML Requests Standard Extension

The request extension consisted in adding a new set of information to provide the context in which the request was conceived. As illustrated by figure 4, this set of information is nested to the new node (*Context*) which is formed by the local from where the request comes (*internal* or *external* to the network in which the system that holds the requested resource is), user role level (*veryHigh*, *high*, *medium*, *low*, *veryLow*), type of machine used to make the request (*desktop* or *mobile device*) and, finally, the type of application protocol (*ssh*, *ftp*, *http*).

```
<Request xmlns="...">
   <Subject>
      <Attribute>
        ...
      </Attribute>
   </Subject>
   <Resource>
      ...
   </Resource>
   <Action>
      ...
   </Action>
   <Environment>
      ...
   </Environment>
   <Context>
      <AccessLocation>internal</AccessLocation>
      <UserRole>veryHigh</UserRole>
      <MachineType>desktop</MachineType>
      <ApplicationProtocol>ssh</ApplicationProtocol>
   </Context>
</Request>
```

Figure 4: An example of the extended XML request.

Thus, it is possible to determine more precisely the factors involved in the request. Using the context from where the request came, we got a set of important information for calculating risk as shown in section 5.2.1.

5.2 The XACML Architecture Extension

The extension of XACML architecture was developed with the addition of a new set of blocks which are highlighted inside the square in figure 5. In this extension, the risk is calculated by the *Request Enforcement Risk* block which receives the execution flow when the PDP block determines that no XACML policy is applicable on the current request. Thus, our proposed architecture *preserves* the original execution of XACML data flow architecture and triggers risk calculus when there is no policy available to address access control to such request. Thus, rather than having to *break the glass* to gain access control, i.e., forcing up the rights and power of a request if needed, our proposed architecture provides a risk-based access control to enable a dynamic chance to access resources available, for example, on a cloud environment.

This architecture avoids the need for foreseeing and describing in policies all contexts in which user can request access to a resource. As there are many contexts to consider, lots of them could be forgotten or omitted by the policy administrator. The risk calculus is a way to decide about access for contexts which were not predicted. Thus, there will always be a guarantee that all contexts will be evaluated either through policies or through risk calculus.

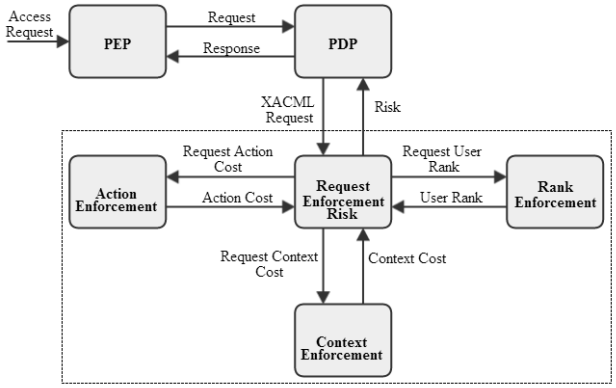

Figure 5: Extended XACML data flow architecture.

The next sections describe how the new blocks work into the proposed extension of XACML architecture. All the values of factors necessary in the next formulas are retrieved from a set of XML files which specify the users ranks, the actions which could be performed and the elements that form contexts.

5.2.1 Request Enforcement Risk Block

This block is responsible for calculating the risk of a request applying formula 1.

$$Risk = (w_1 * ContextCost + w_2 * ActCost) - w_3 * RankCost \quad (1)$$

This formula considers the cost of three factors: *context* (*ContextCost*) of the request, *action* (*ActCost*) to be taken over the desired resource and *user rank* (*RankCost*) of the subject invoking the access request. These factors are calculated by the blocks *Context Enforcement*, *Action Enforcement* and *Rank Enforcement*, respectively. To each factor a weight (w_1, w_2 and w_3) is assigned expressing the importance degree of the factor in the risk calculus.

The main idea behind this formula is to sum the costs associated to context and action and over the result subtract the user rank value. For an access request to be allowed it is necessary the comparison of the risk value returned by formula 1 with a threshold, such that, the risk must be inferior than the threshold. The threshold is defined by formula 2. Thus, it avoids the need for an administrator to set a comparison value. Formula 2 takes into account the average of the n possible contexts of requests (different combinations of its elements), the m possible actions which could be specified in a given request and performed over a resource and the k users ranks of individuals which could perform access request. The risk of a request must be less than the threshold.

$$Threshold = (w_1 * \frac{\sum_j^n ContextCost_j}{n} + w_2 * \frac{\sum_j^m ActCost_j}{m}) - (w_3 * \frac{\sum_j^k RankCost_j}{k}) \quad (2)$$

If in a given context at time t an user does not have access to a resource, he may have access to such resource in the same context at time t' , with $t < t'$, if the value of his user rank increase enough to make it possible.

5.2.2 Context Enforcement Block

This block is responsible for calculating the cost associated with the context of the access request. It obtains the values related to the context from the extended request presented in section 5.1 and applies formula 3.

$$(w_4 * accessLocation + w_5 * machineType + w_6 * appProtocol + (w_7 * userRole) \quad (3)$$

A weight (w_4, w_5, w_6 and w_7) is assigned to each context term expressing their importance degree. The values of such terms are obtained from a XML configuration file as exemplified in figure 6.

```
<context>
  <accessLocation>
          <internal>1</internal>
          <external>5</external>
  </accessLocation>

  <userRole>
          <veryHigh>1</veryHigh>
          <high>2</high>
          <medium>3</medium>
          <low>4</low>
          <veryLow>5</veryLow>
  </userRole>

  <machineType>
          <desktop>3</desktop>
          <mobile>5</mobile>
  </machineType>

  <appProtocol>
          <ssh>1</ssh>
          <ftps>1</ftps>
          <http>3</http>
  </appProtocol>
</context>
```

Figure 6: XML file to associate values to context terms.

According to our view, the higher the security a term expresses *less* will be its value, while if the term expresses less security its value will be *bigger*. For example, in a range from 1 to 5, a very important user role (*veryHigh*) takes 1, while a user with minor role (*veryLow*) takes 5. Thus, contexts considered safe (those having context elements with small values) have a low score and insecure contexts (those having context elements with big values) have high score. The choice of the range and the value to be mapped to each term must be defined by an administrator who can evaluate how important a context term is.

5.2.3 Action Enforcement Block

This block is responsible for calculating the cost associated to the action to be taken over the desired resource. To achieve this goal, the block *Risk Assessment* proposed in [4] and presented in section 4 was adapted and implemented in our architecture.

Let A be the set of user actions and O_{a_i} be the set of outcomes of action $a_i \in A$. For each result $o_{j_{a_i}} \in O_{a_i}$, its risk is calculated considering the characteristics of *availability*, *integrity* and *confidentiality* according to the formulas 4, 5 and 6, respectively.

$$R_D(o_{j_{a_i}}) = P(o_{j_{a_i}}) * C_D(o_{j_{a_i}}) \quad (4)$$

$$R_I(o_{j_{a_i}}) = P(o_{j_{a_i}}) * C_I(o_{j_{a_i}}) \quad (5)$$

$$R_C(o_{j_{a_i}}) = P(o_{j_{a_i}}) * C_C(o_{j_{a_i}}) \quad (6)$$

Where, $C_D(o_{j_{a_i}})$, $C_I(o_{j_{a_i}})$ and $C_C(o_{j_{a_i}})$, in this sequence, correspond to costs of outcomes $o_{j_{a_i}}$ in terms of *availability*, *integrity* and *confidentiality* and $P(o_{j_{a_i}})$ refers to the probability of $o_{j_{a_i}}$ occurrence.

Formulas 7, 8 and 9 determine the risk values over the previously defined equations, where $p = |O_{a_i}|$.

$$R_{VD}(a_i) = \sum_{j}^{p} R_D(o_{j_{a_i}}) \quad (7)$$

$$R_{VI}(a_i) = \sum_{j}^{p} R_I(o_{j_{a_i}}) \quad (8)$$

$$R_{VC}(a_i) = \sum_{j}^{p} R_C(o_{j_{a_i}}) \quad (9)$$

Finally, the overall risk associated with an action a_i is given by formula 10, where the possible outcomes of an action are specified considering availability, integrity and confidentiality aspects.

$$R(a_i) = w_8 * R_{VD}(a_i) + w_9 * R_{VI}(a_i) + w_{10} * R_{VC}(a_i) \quad (10)$$

The XML file of figure 7 exemplifies the format in which an action must be written. In this example, the allowed action is *read* and *unavailable* is one of its possible outcomes.

```
<actions>
  <read>
    <outcomes>
      <unavailable>
        <availability>
           <probability>0.4</probability>
           <impact>10</impact>
        </availability>
        <integrity>
           <probability>0.3</probability>
           <impact>0</impact>
        </integrity>
        <confidentiality>
           <probability>0.4</probability>
           <impact>1</impact>
        </confidentiality>
      </unavailable>
    </outcomes>
  </read>
</actions>
```

Figure 7: Format of an action.

5.2.4 Rank Enforcement Block

The *Rank Enforcement* block is responsible for returning the *user rank* from the entity that request a resource, where rank express numerically the entity behavior over time. Entities with good behavior have high ranks, while bad behavior implies in low rank values.

The rank allows a dynamic risk calculus, as time progresses, the rank value changes and it impacts the result of the risk calculus. If in a given context in time t a user did not have access to a resource, he could have access to such resource in the same context at a time t', if his rank value increases enough.

In this work, the context of request is dynamically determined at runtime and then the permission is decided. However, a certain context may repeat over time, then the decision for such context will be always the same. With the introduction of rank, even if the context repeat over time, the decision may be different.

This block is responsible just for maintaining and returning the ranks values. The monitoring of user behavior as well as the assignment of rank values are not addressed in this work. It is assumed that the system which uses this extension is responsible for implementing an user behavior monitoring. A reference that deals with monitoring of user behavior and rank inference is [9].

6. TESTS AND RESULTS

A prototype was developed in a cloud environment to evaluate access control policies dynamically, using the XACML extension proposed in this work. Figure 8 presents the implemented structure. The main tools used were: XML language for defining data and requests; Amazon EC2 cloud to host a client application and a server application (which implements our proposal for evaluating the risk-based access control); Java EE language; Tomcat application server; and Web services to communicate between different parts of the application.

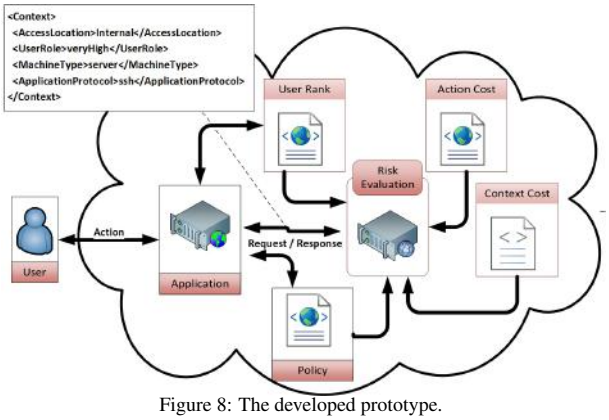

Figure 8: The developed prototype.

Suppose the following tests were performed in a hospital in which doctors and nurses have access to patients' medical records available on the hospital's cloud environment. Table 1 shows which kind of user role doctors and nurses play at the hospital according to their service time. Doctors play user role as *very high* when their service time is superior than 3 years, otherwise their user role is *high*, while nurses play user role as medium when their service time is superior than 3 years, *low* when their service time is between 1 and 3 years, or *very low* when their service time is less than 1 year.

Staff	Service Time	User role
Doctor	$>$ 3 years	very high
	$\leq$ 3 years	high
Nurse	$>$ 3 years	medium
	$\geq$ 1 and $\leq$ 3 years	low
	$<$ 1 year	very low

Table 1: User roles to doctors and nurses.

The tests performed access requests to patients' medical records in different contexts for which no policy were predicted. Thus, the prototype realized risk calculus to decide about the permissions. In such tests some contexts were repeated over time in order to verify how much impact the varying of user rank value caused in the risk calculus. Since we consider the degree relevance of *availability*

bigger than *integrity*, which by its turn, is bigger than *confidentiality*, then the weights of formula 10 were $w_8 = 0.5$, $w_9 = 0.3$ and $w_{10} = 0.2$ and *read* (see section 5.2.3) was the action taken over the required resources. Note that such weight values can be set to any value according to what the administrator thinks is more appropriate.

The values obtained for calculating the contexts are those showed in the section 5.2.2 and 0.25 was assigned to weights of formula 3. We consider the factors which compose formula 3 have the same relevance degree. It was determined that a user rank can assume a value in the range [0,10] and the average of users rank was 6. Such range can not be too large because a very high user rank can influence the risk calculus to turn an access request acceptable even in unsafe contexts. In formula 1 the weights of context cost and action cost were assigned with the same value, such that, $w_1 = 0.45$ and $w_2 = 0.45$. The weight of user rank was assigned with a low value, $w_3 = 0.1$, so it is not a determinant factor in the risk calculus but it is a complement which may assist the calculus to give access in contexts which repeat over time. Moreover, the cost associated to action *read* (see figure 7) is 2.06. Finally, using formula 2 we obtained a threshold value equals to 1.62.

Test 1

At time t, in which test 1 was performed, a nurse whose service time is less than 3 years and bigger than 1 year and user rank is equal to 4 requests an access in which context is illustrated by figure 9.

```
<Context>
    <AccessLocation>external</AccessLocation>
    <UserRole>low</UserRole>
    <MachineType>mobile</MachineType>
    <ApplicationProtocol>http</ApplicationProtocol>
</Context>
```

Figure 9: Context of test 1.

The risk value is calculated as follows (see formula 1):

$$Risk = (0.45 * 4.25 + 0.45 * 2.06) - 0.1 * 4$$

The request is rejected because the risk value is 2.43 that is bigger than the threshold 1.62. This happened because the combination of context elements are mostly insecure, that is: a request access from an external network is more dangerous than when it is from the internal network where the requested resources are located; using a mobile device to make a request is not much safe because this kind of device is generally not equipped with security mechanisms and it runs the risk of being stolen; HTTP is a protocol with very security limitations; for many reasons an employee (a nurse in this case) with low service time has very limited access to resources of an enterprise.

Test 2

The same nurse from test 1 tries a new request access at time t', with $t < t'$, such that her service time is now bigger than 3 years and her user rank is equal to 4. The new context, in which test 2 was performed, is illustrated by figure 10.

The value of the risk is calculated as follows:

$$Risk = (0.45 * 4 + 0.45 * 2.06) - 0.1 * 4$$

The request is also rejected because the risk value is 2.33 that is bigger than the threshold 1.62. Even with the user role changing from low to medium the request access is rejected because the combination of context terms are still mostly insecure.

```
<Context>
    <AccessLocation>external</AccessLocation>
    <UserRole>medium</UserRole>
    <MachineType>mobile</MachineType>
    <ApplicationProtocol>http</ApplicationProtocol>
</Context>
```

Figure 10: Context of test 2.

Even if the user rank become 10 (the maximum allowed value to users ranks according to our configurations) and a new request was performed for this context the request would be rejected as well (the risk would be 1.73). This test shows that a good user rank should not allow access in high-risk contexts. For this request being accepted, the context elements must change to safer ones and the user rank must increase.

Test 3

The same nurse from the above tests requests a new access at time t'', with $t' < t''$, such that her user rank is equal to 4 and the context, in which test 3 was performed, is illustrated by figure 11.

```
<Context>
    <AccessLocation>internal</AccessLocation>
    <UserRole>medium</UserRole>
    <MachineType>desktop</MachineType>
    <ApplicationProtocol>http</ApplicationProtocol>
</Context>
```

Figure 11: Context of test 3.

The value of the risk is calculated as follows:

$$Risk = (0.45 * 2.5 + 0.45 * 2.06) - 0.1 * 4$$

In this case, the context elements are safer than the ones in tests 1 and 2, however it was not possible to release the access to the required resource because the risk value is equal to 1.65. Note that changing the context elements decreased the risk value and the request was almost accepted.

Increasing the user rank to 6 make possible the access to the required resource since the new risk value becomes 1.45. Even if the user rank become 4,32 (users ranks could decrease over time according to their behaviors) and a new request was performed for this context the request would be still acceptable (the risk would be 1.62).

Test 4

At time t, in which test 4 was performed, a doctor whose service time is less than 3 years and user rank is equal to 4 requests an access in which the context is illustrated by figure 12.

```
<Context>
    <AccessLocation>external</AccessLocation>
    <UserRole>high</UserRole>
    <MachineType>desktop</MachineType>
    <ApplicationProtocol>http</ApplicationProtocol>
</Context>
```

Figure 12: Context of test 4.

The risk value is calculated as follows:

$$Risk = (0.45 * 3.25 + 0.45 * 2.06) - 0.1 * 4$$

The request is rejected because the risk value is 1.99 that is bigger than the threshold 1.62. This happened because the combination of context elements are mostly insecure. Note that, excepting

the user role, the context elements of this test are the same of test 1. However, in this test the risk is less than the risk of test 1 because a doctor's user role is more important than a nurse's user role.

Test 5

The same doctor from test 4 whose user rank increased to 7 tries a new request access at time t', with $t < t'$, the context, in which test 5 was performed, is illustrated by figure 13.

```
<Context>
    <AccessLocation>external</AccessLocation>
    <UserRole>high</UserRole>
    <MachineType>desktop</MachineType>
    <ApplicationProtocol>http</ApplicationProtocol>
</Context>
```

Figure 13: Context of test 5.

The risk value is calculated as follows:

$$Risk = (0.45 * 2.5 + 0.45 * 2.06) - 0.1 * 7$$

The request is accepted because the risk value is 1.35 that is less than the threshold 1.62. This test shows that a good rank could allow access in medium-risk contexts.

Test 6

At time t, in which test 6 was performed, another doctor whose service time is bigger than 3 years and user rank is equal to 7 requests an access in which context is illustrated by figure 14.

```
<Context>
    <AccessLocation>internal</AccessLocation>
    <UserRole>veryHigh</UserRole>
    <MachineType>desktop</MachineType>
    <ApplicationProtocol>http</ApplicationProtocol>
</Context>
```

Figure 14: Context of test 6.

The risk value is calculated as follows:

$$Risk = (0.45 * 2 + 0.45 * 2.06) - 0.1 * 7$$

Since the context elements are secure elements and user rank is above average of ranks, the request was readily accepted with value equals to 1.13.

Test 7

The same doctor from test 6, now with user rank equals to 10, tries a new request access at time t', with $t < t'$, in which context is illustrated by figure 15.

```
<Context>
    <AccessLocation>internal</AccessLocation>
    <UserRole>veryHigh</UserRole>
    <MachineType>desktop</MachineType>
    <ApplicationProtocol>ssh</ApplicationProtocol>
</Context>
```

Figure 15: Context of test 7.

The risk value is calculated as follows:

$$Risk = (0.45 * 1 + 0.45 * 2.06) - 0.1 * 10$$

The risk value is 0.38, so the request was accepted. This context is formed by the safest elements, so there is almost no risk in accepting this request.

7. CONCLUSION

Cloud computing has a large number of challenges related to security of resources. In this sense, this work contributed in defining a way to control access to resources on the cloud, by extending the XACML language to support the RAdAC model, presenting a quantitative and concrete form of risk calculus, considering the context, impact on confidentiality, integrity and availability of resources and the introduction of the user rank. The introduction of user rank allowed a dynamic risk calculus such that users who did not have access to a resource in a given time t could have it at time t' if their ranks increase enough for that.

The implemented model is easily handled, the administrator just need to change the XML files and the weights of the formulas as needed. Moreover, this model is flexible, so it may be easily adapted or extended. In a next step we intend to focus our efforts in determining formal means to quantify the RAdAC *operational need* and implement it in our XACML architecture.

The work of Meeta Sharma et. al (2012) considers only the characteristics of confidentiality, integrity and availability for risk calculus in cloud environments, but let to the administrator or user the task of definition of risk policies. Chen Chen et. al (2010) proposed a set of modules to continuously control the risk calculus. If access is released, the risk is recalculated from time to time, considering changes in the data context throughout the session. However, the authors do not present a concrete formula of how risk should be calculated. Liang Chen (2013) extended the XACML language to support the RAAC model. The main variables associated with entities involved in the access request may be expressed as attributes. However, the authors do not present how the risk is in fact calculated.

These works are concerned in presenting new components, how they work, test scenarios and factors that should be considered for risk calculus, although do not present how exactly the risk is calculated. Some future works can be developed to: a) test the application using a real case study to validate our approach; b) add intelligent components to adapt user ranks according to history of user actions; c) implement the RAdAC's operational need; d) implement an automatic and smarter way of defining the weighs of the previous formulas.

References

[1] Security for cloud computing: 10 steps to ensure success. Technical report, 2012.

[2] Chen Chen, Weili Han, and Jianming Yong. Specify and enforce the policies of quantified risk adaptive access control. In Weiming Shen, Ning Gu, Tun Lu, Jean-Paul A. Barthès, and Junzhou Luo, editors, *CSCWD*, pages 110–115. IEEE, 2010.

[3] Liang Chen, Luca Gasparini, and Timothy J Norman. Xacml and risk-aware access control. 2013.

[4] Nguyen Ngoc Diep, Sungyoung Lee, Young-Koo Lee, and Heejo Lee. Contextual risk-based access control. In Selim Aissi and Hamid R. Arabnia, editors, *Proceedings of the 2007 International Conference on Security & Management, SAM 2007, Las Vegas, Nevada, USA, June 25-28, 2007*, pages 406–412. CSREA Press, 2007.

[5] Daniel Ricardo dos Santos, Carla Merkle Westphall, and Carlos Becker Westphall. A dynamic risk-based access control architecture for cloud computing. In *Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium NOMS 2014*. IEEE/IFIP, IEEE Press, May 2014.

[6] Thomas Erl and Zaigham Mahmood Ricardo Puttini. *Cloud Computing: Concepts, Technology and Architecture*. The Prentice Hall Service Technology Series from Thomas Erl. Pearson Education, 2013.

[7] Doudou Fall, Gregory Blanc, Takeshi Okuda, Youki Kadobayashi, and Suguru Yamaguch. Toward quantified risk-adaptive access control for multi-tenant cloud computing. In *Proceedings of the 6th Joint Workshop on Information Security (JWIS 2011), Kaohsiung, Taiwan, August 2011*, 2011.

[8] Bassam Farroha and Deborah Farroha. Challenges of operationalizing dynamic system access control: Transitioning from abac to radac. In *IEEE International Systems Conference*, pages 1–7, march 2012.

[9] Giorgos Giannopoulos, Ulf Brefeld, Theodore Dalamagas, and Timos Sellis. Learning to rank user intent. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 195–200, New York, NY, USA, 2011. ACM.

[10] Heras. http://www.herasaf.org/heras-af-xacml.html, 2012. [Accessed in 12/04/2015].

[11] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[12] OASIS. extensible access control markup language version 3.0 oasis standard, 2013.

[13] JASON Program. Horizontal integration: Broader access models for realizing information dominance. Technical report, MITRE Corporation, 12 2004.

[14] Meeta Sharma, Yan Bai, Sam Chung, and Lirong Dai. Using risk in access control for cloud-assisted ehealth. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 1047–1052, 2012.