



POLITECNICO DI TORINO

Corso di Laurea in Communications and Computer Networks Engineering

Tesi di Laurea

# **A Context-Aware Risk-Based Authorization System**

**Relatori**

Prof. Antonio Lioy

Ing. Andrea Atzeni

Emanuele CELORIA

ANNO ACCADEMICO 2017-2018



# Summary

Despite the great success that the role-based paradigm has had and still has within the access control topic, there exist scenarios that, due to their dynamic intrinsic nature, present an inherent uncertainty when taking authorization decisions and for which it may be an hard task to predict which kind of authorization permissions have to be granted to users, and which not, especially when the working environments may sensibly vary. It follows that, for those scenarios, a static approach, as the one proposed by role-based authorization systems, may not be the preferable one. With the advent of the new millennium some proposals have been done in order to address the problem with more dynamic implementations of access control systems, able to exploit the notion of risk in order to take authorization decisions; anyway from the implementation point of view very few attempts have been made.

In this thesis work a risk-based authorization system, able to exploit the characterization of environments in order to be context-aware, is proposed and implemented.

The system has been built up by integrating CAIRIS and SAFAX tools. The first is a security and usability requirements management tool, able to deal with risk informations related to assets, threats, vulnerabilities and environments, the second is a XACML-based architectural framework tailored to the development of extensible authorization services.

The SAFAX service will exploit the risk notions, present in CAIRIS, when examining an authorization request against a policy. The great advantage that the exploitation of the CAIRIS service brings into the usage of the whole system is to allow a preliminary configuration phase, during which a user is able to create a descriptive and visual risk model, specific for each environment which characterize a certain project and from which risk scores will be automatically derived.

In order to show the potential of the developed system, three usage scenarios are going to be presented. In the first, it is presented how the risk-based authorization mechanism would positively impact authorization decisions taken from the information system of a water company, which has often to manage the access to critical resources, in order to carry out unforeseeable interventions for the correct functioning of the infrastructure. In the second, an healthcare scenario is taken into consideration, introducing the risk-based authorization system in order to manage the access to sensitive resources in different working situations. Finally, in the third scenario, the authorization system proposed is applied to a grid-based collaborative network for clinical researchers, where sensitive informations about patients and diseases have to be accessed from different network communities.

# Contents

<b>1</b>	<b>Introduction</b>	7
<b>2</b>	<b>State of the Art Analysis</b>	10
2.1	Towards the definition of a mathematical and architectural model for computing risk . . . . .	10
2.2	Introducing benefit analysis process . . . . .	16
2.3	Supporting the new Cloud Computing paradigm . . . . .	16
<b>3</b>	<b>Requirements for a Risk-Based Access Control Framework</b>	21
3.1	Architectural Requirement Analysis . . . . .	21
3.1.1	Risk-Based Approach . . . . .	21
3.1.2	Trust & Risk-Based Approach . . . . .	24
3.1.3	Obligations Enforcement . . . . .	26
3.2	Security Implications . . . . .	26
<b>4</b>	<b>Analysis of the Tools: CAIRIS and SAFAX</b>	30
4.1	CAIRIS . . . . .	30
4.1.1	Introduction . . . . .	30
4.1.2	Baseline Architecture and Goals . . . . .	31
4.1.3	Novelties . . . . .	34
4.2	SAFAX . . . . .	37
4.2.1	Introduction . . . . .	37
4.2.2	Baseline Architecture and Goals . . . . .	38
4.2.3	Novelties . . . . .	40
<b>5</b>	<b>Integration</b>	42
5.1	High Level Implementation . . . . .	42
5.2	Architecture . . . . .	43
5.3	Performance Considerations . . . . .	46

<b>6 Usage Scenarios</b>	51
6.1 Scenario 1: ACME Water	51
6.1.1 Situation 1:	52
6.1.2 Situation 2:	53
6.2 Scenario 2: Healthcare facility	53
6.2.1 Situation:	55
6.3 Scenario 3: NeuroGrid	55
6.3.1 Situation:	57
<b>7 Performance Results</b>	58
<b>8 Conclusions</b>	64
8.1 Lessons learnt and open issues	64
8.2 Future works	67
<b>Bibliography</b>	71
<b>A Developer Manual</b>	77
A.1 “cairis”module	77
A.1.1 /nl/tue/sec/cairis/db	77
A.1.2 /nl/tue/sec/cairis/engine	81
A.1.3 /nl/tue/sec/cairis/impl	82
A.1.4 /nl/tue/sec/cairis/util	83
A.1.5 /nl/tue/sec/cairis/ws	87
A.1.6 The other “cairis”subfolders	88
A.2 “sfx”module	88
A.2.1 WebContent/main.html	89
A.2.2 WebContent/js/sfxmain.js	89
A.3 “sfxservice”module	89
A.3.1 package nl.tue.sec.safax.sfxbe.db	89
A.3.2 package nl.tue.sec.safax.sfxbe.impl	90
A.3.3 package nl.tue.sec.safax.sfxbe.ws	90
A.4 db_risk_tables folder	91
A.5 example_policies_requests folder	91

<b>B</b>	<b>User Manual</b>	102
B.1	Installation Guide	102
B.1.1	Tomcat installation	102
B.1.2	Safax repository download via SVN	103
B.1.3	MySQL	103
B.1.4	Execution Environment	104
B.2	User Guide	107
B.2.1	Advanced Functionalities	107

# Chapter 1

## Introduction

The adoption of adequate measures to guarantee the security of information systems and resources deployed on those systems should, nowadays, be a solid and well accepted procedure, not only within big organizations, but in more contained scenarios too. To this purpose, access control systems have been widely adopted in order to deal with all sort of applications, from financial to healthcare and in many business scenarios.

An access control system should be able to determine, after a proper authentication procedure, the activities, associated to the permissions, allowed to a legitimate user. For this reason, in addition to an authentication procedure, a proper authorization mechanism should be present. A possible definition for “authorization” is given by the RFC4949 [68], which defines it as “a process for granting approval to a system entity to access a system resource”.

One of the most prominent solutions characterizing authorization mechanisms is identified by the role-based paradigm. The interest in the Role-Based Access Control’s (RBAC) topic began to grow in the last decades of the 20th century, formalized as a model for the first time in 1992 by Ferraiolo and Kuhn [22], until reaching a great relevance in 2004 when it was adopted as American National Standard (ANSI-INCITS 359-2004) by the American National Standards Institute [35]. The basic idea of the RBAC model is that users and permissions are associated with roles, in such a way of avoiding the definition of customized policies for each single user, which is not a negligible issue within a big organization. In other words, a role provides a convenient way of associating a group of users with a set of permissions. Moreover, roles can be defined hierarchically, in such a way to let the possibility, for high level roles, to inherit permissions from the lower level ones and, in this way, further ease the definition of policies.

Despite the great success the RBAC model obtained in a lot of applications, it would be too simplistic to think that existing role-based authorization mechanisms were sufficient to provide the appropriate level of security and usability needed to cope with all kinds of situations and environments, especially nowadays that the world wide society relies this much on IT infrastructures and systems.

Information systems security is probably one of the hottest topic at the moment, but, in order to be accepted, has surely to be balanced with usability measures. The problem was already addressed by some studies in 1999 in [3], where authors give evidence to the fact that often the way in which security mechanisms are implemented does not take into account human factors, causing, in this way, a more difficult interaction between users and systems, that can bring to an improper use of the system itself, compromising its inner security. The same issues are identified in [77], where the author explains how integrating a user-centred approach, when designing security mechanisms, could be of great help in order to avoid conflicts between security and usability implications. Indeed, while security focuses on users’ and systems’ protection, usability brings more attention on the quality of the user-system interaction. When security measures become too tight, the usability of systems is mined and vice versa the risk of a too “user friendly” system is to

lack of appropriate security measures. Flexibility in finding a trade off between the two concepts is then fundamental.

The real problem in existing authorization mechanisms is that they allow to specify policies that rely on hard coded static decisions, which make them inflexible and difficult to adapt to changing circumstances. Either someone, who is trying to perform an action, has the appropriate credentials, that provide a sufficient trust level to carry out the task, or there is no way to accomplish the task's goal. This can be appropriate for particular scenarios, in which we want to be statically restrictive and conservative when evaluating policies, but, on the other end, can be also very limiting in dynamic environments, where authorization decisions can substantially vary depending on the specific context they are facing. In those cases, it is not an easy task to define access control policies able to provide both a sufficient protection for the resources and an appropriate functionality level to let users accomplish their tasks. The risk, indeed, is to have a system which is either too permissive towards possible risky entities or too restrictive towards entities that, in a specific context, should be trusted. In this latter case, the security issues that could arise from such an unusable system cannot be neglected, indeed they could create unpredictable situations where users decide to carry out the task via undocumented means. It follows that, sometimes, even though a controlled amount of risk associated to an authorization permission is present, and if the benefits, resulting from the accomplishment of the task, related to that permission, outweigh those risks, it is better to grant the authorization than steadily blocking the system. Let's think for example to an healthcare scenario, in which urgencies to access sensitive patient's informations could occur either to high profile members of the organization, like doctors, which would probably be allowed to access those kinds of informations, but also to less trusted members, like nurses or simple medical assistants, that normally wouldn't be allowed to examine those informations but exceptionally they could find themselves involved in emergency situations, requiring it. In those cases, even the applying of a simple medical precaution, rendered possible by the examination of sensitive clinical records of the patient, could result in an invaluable benefit for the patient itself, whose life could be saved.

The motivation underlying the arising of this thesis work comes from the idea of revising the way in which authorization decisions are taken. The answer proposed to the identified limitations, characterizing the traditional RBAC model, relies on the idea of considering, during the policy evaluation phase, values of risk, which are evaluated runtime, allowing the possibility to take authorization decisions that can slightly change in the time and by varying the context in which those risks are considered, thus addressing the lack of flexibility identified in the RBAC model. For this reason, a risk-based authorization system, able to dynamically adapt to situations and scenarios, proposes itself as the most suitable solution when dealing with complex and articulated situations where unexpected circumstances and exceptions are on the daily agenda.

The way risk has been treated in literature can slightly vary among models, in [18], for example, risk is computed as the product between the value of information, identified by the damage sustained if this information were disclosed in an unauthorized manner, and the probability of unauthorized disclosure; in [28] Faily, instead, found its model above the definition of risk as the product of the threat likelihood, the severity of the vulnerability, and the risk impact to the threatened assets. Anyway it is usually accepted to consider risk as the expected damage.

The purpose of a dynamic risk-based solution, though, is not only to improve the usability of an authorization system but also to eventually counteract the credential abuse problem, which represents any unapproved or malicious use of organizational resources. According to the 2017 Data Breach Investigations Report [1] of Verizon, the 14% of tactics, used in data breaches, involved privilege misuse and the 25% of breaches has been carried out by internal actors, showing us how critical is, today, for a company, to be appropriately protected not only from the external world but from the inside as well. A breach is defined as an incident that results in the confirmed disclosure (not just potential exposure) of data to an unauthorized party.

The report shows that Public Administration and Healthcare industries are the mostly touched by

the privilege misuse problem; the 71% of data compromised involved personal informations and medical records, which have been targeted mainly for financial crimes, such as identity theft or tax-return fraud and occasionally just for gossip value. A solid 24%, instead, involved sensitive internal data like sales projections and intellectual property, that could be used either to start up a competing company or to take to a new employer. These numbers are significant because allow us to understand the huge impact that a naive and not up-to-date access control mechanism could have inside an organization, with the possibility of resulting in huge damages for the organization itself and also for the people which rely on it. Ahead in the thesis some scenarios of usage, for the risk-based authorization solution proposed, are going to be presented and one of the focuses is going to be precisely on the healthcare environment.

The rest of the thesis is organized as follows: in Chapter 2, we revise the state of the art for what concerns Risk-Based Access Control systems, explaining how the topic is dealt with and which are, currently, the proposed solutions. In Chapter 3, we summarize the basic necessary requirements, identified in the previous chapter, to build a risk-based access control framework and we identify those supplementary requirements necessary to let the system evolve with the introduction of trust and obligation concepts. In Chapter 4 we introduce the two tools that have been the main focus of this thesis work and that have been integrated in order to obtain the proposed risk-based authorization system, highlighting their main features, their goals and the novelties that bring in respect to similar already existing tools. In Chapter 5, we present the integration between the two tools, highlighting the final architecture and its way of functioning, moreover some expectations about performances of the system are outlined. In Chapter 6, we present some usage scenarios for the system, in order to show some examples and to inspire the reader about the possibilities this risk-based authorization system has to be used in real cases. Chapter 7 presents the results obtained from a performance evaluation of the system. Finally, in Chapter 8, we draw conclusions, lesson learnt will be outlined, performing a critical thinking of the work done and presenting some future work that could be carried on in order to enhance the developed system.

## Chapter 2

# State of the Art Analysis

In this chapter it is going to be revised the state of the art of risk-based access control systems, an approach to deal with access control that started to be investigated with the advent of the new millennium and which does not rely on the well-known Role-Based mechanisms but it exploits the concept of risk in order to take authorization decisions.

Slightly different proposals have been done in the time, some of them are more technology specific, like the ones referring to Cloud Computing scenarios, some others are more focused in presenting the mathematical approach to compute the risk, others instead mainly focus on the architecture over which implement their access control idea. Finally, some other proposals find their main applicability in specific contexts particularly related to confidentiality problems, although the idea of the access control implementation, brought by this thesis' work, is to implement a solution able to cope with a wider range of risks, that can arise when asking for certain authorization permissions, in diverse contexts.

### 2.1 Towards the definition of a mathematical and architectural model for computing risk

One of the first approaches to risk-based access control that has been proposed in the literature is Fuzzy MLS [18]. The basic premise of this approach is that in access control decisions a certain measure of uncertainty is always present due to difficulties in quantifying risk, which is defined as the product between the value of the information and the probability of unauthorized disclosure. The value of the information is given by a rough estimate that can be done by the organization, while the probability of unauthorized disclosure is trickier and it's given by a combination of probability values obtained by applying a *sigmoid* function. According to this model, labels are assigned to subjects and objects. A subject's label is composed by its sensitivity level, which reflects the degree of trust of the subject, and the categories set, representing categories of objects for which the subject has a legitimate *need-to-know*. An object's label, as well, is composed by its sensitivity, which should reflect the magnitude of damage if an unauthorized access occurs, and by the categories set the object belongs to. Each access is characterised by a *gap* between subject's label and object's label that points to a precise point in the risk scale. This risk scale has a soft and an hard boundary; if the risk is below the soft boundary, access is granted, else if the risk is above the hard boundary the access is denied. Between the soft and the hard boundary there are certain bands of risk, each one associated to a mitigation measure. Mitigation measures correspond to obligations that subjects need to accomplish in order to reduce the risk associated to their request. In Fuzzy MLS model, these mitigation measures rely on a *Line of Risk Credit* approach (similar to a credit card system). Each user has a credit of risk, every time a user accesses

an information whose risk is within the region of bands of risk, the difference between the risk and the soft boundary is charged against the user's risk credit.

This approach could be probably reasonable in environments where users are considered sufficiently trustworthy, otherwise for a malicious user could be quite simple to have access to sensitive resources. Think about a user which is thinking of leaving the company, he could do a significant amount of damage on purpose by using all his risk credit.

Another, among the very first proposals, has been the one presented by McGraw, called RAdAC [50], which basically identifies three modules: a security risk module, an operational need module and an access decision module. The security risk module is in charge of computing the risk associated to the request (determined as a composite risk computed over several factors); the operational need module is in charge of quantifying (not just a simple binary classification) the need-to-know associated to the person performing the request; the access decision module receives the results of the two previous modules and return an access decision (permit/deny). If the risk is above a maximum threshold, request is denied, regardless of the operational need; if the risk is in the region between a normal and the maximum threshold, then we look at the operational need: if this is above a critical need threshold (set by the policy) then this can override the risk, instead, if the risk is below the normal threshold then if the operational need is at least above a minimum need-to-know threshold then the access is granted. In RAdAC, the author only describes an high level module to be integrated within a dynamic access control model but does not explain a risk model, indeed no details on how to measure quantitatively the risk and the operational need are given.

Fuzzy BLP system [55] was derived from the approach previously described in [18]. Even here we have categories to which objects are assigned, each category has different properties and each property has a score. An object's score is computed as the sum of the scores of all its categories' properties. A security label with four entries (*unclassified*, *classified*, *secret*, *top secret*) is assigned to each document and to each one of the entry corresponds a membership function, which computes the degree (within [0,1]) of membership of the document to each one of the label's entries. Same process is performed similarly for the subject. Then, relying on a BLP risk inference rules table, which takes into account antecedence parameters (object's and subject's label) and consequence parameter (predicted damage), the final risk is computed. The idea then evolves similarly to Fuzzy MLS [18], using a credit card based approach with the difference that the user's risk credit, now called access tokens, is charged according to obligations access quota. The idea of associating access quotas to obligations, and not directly to resources, is due to the fact that the risk, for an object, varies depending on the context and the subject trying to accessing it. Thus the idea of assigning access quotas, as a sort of "price", to obligations, even though there is no further explanation of what an obligation should consist in and why should be easier to do so. Anyway, when accessing a certain object, obligations need to be fulfilled. The general information flow, proposed by the paper, to support an access control decision is shown in figure 2.1. The other difference that the authors claim is that Fuzzy BLP provides a general methodology to implement customised risk-based access control rules without using a particular function (like the *sigmoid* function used in Fuzzy MLS) to compute the risk.

An interesting, although not very much developed, work has been carried on by Ahmed and Zhang with CRAAC [4]. The model is able to identify requesters and resources; each requester is associated to a RLoA (Requester Level of Assurance), computed on several parameters (authentication method, trustworthiness of access location, channel security, type of intrusion detection system). The RLoA is computed as a function of the different parameters based on their relationship (elevating relationship or weakest-link relationship). Resources are instead classified in groups, according to their risk, which have a OLoA (Object Level of Assurance). The paper, anyway, does not present a clear mathematical model to compute this risk, it's an a-priori decision of which group an object fits better in. Access is granted if  $RLoA \geq OLoA$ .

There is no concept of bands of risk and mitigation measures in this approach, but, since requesters

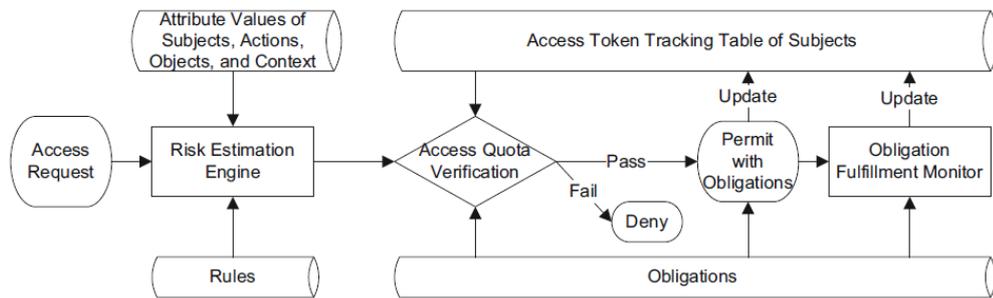


Figure 2.1. The Fuzzy BLP method proposed to support a Risk-Based access control [55].

(users) are not classified in predefined categories, the dynamic of the system is improved and approximates the reality, indeed, now, the trustworthiness of the requester is computed runtime by looking at the specific context. The risk, anyway, remains still quite static, being a predefined parameter.

A slightly different approach is the one proposed in [23], where MFEP (Multifactor Evaluation Process) is described. Risks are accounted on three basic units: availability, integrity, confidentiality. For each possible action, different possible outcomes are defined a-priori. Each outcome has a cost and a probability of occurring. By using the formula  $Risk = Cost * Probability\ of\ occurrence$ , for each outcome associated to the action is computed a risk, separately in terms of availability, integrity and confidentiality. Then, for each one of these basic units, all the risks associated to the outcomes are summed up and the final risk is computed by performing the weighted arithmetic average on those values. The same process is computed both for the solution of accepting and the solution of rejecting, the two resulting values are compared and the solution with minor risk is chosen.

Adopting this approach raises the need for a detailed list of all possible outcomes of an action. This is something not so trivial to do, since it's very hard to be able to foresee all possible events that could result from an action. Anyway, the approach used is very simple and linear and no complex computations are needed. Probably, in a simple environment, in which most of the time outcomes does not go far away from the few ones that can be predicted, it could work well, however, if the situations involve more complex scenarios, the adoption of a more advance Security Requirement Analysis should be considered.

In [76] has been described a risk-based approach based on *need-to-know*. The same concept was introduced in RAdAC model [50], but in that case it was considered as a mitigating factor of the risk computed. This is also different from the majority of previous described models in which risk is evaluated either as a function of cost (impact [23]), or of the trustworthiness of the user ([78]) or of the sensitivity of the data requested ([18],[55]). In particular, this approach suits well environments in which the confidentiality of the data is the major security aspect to take into consideration, such as Healthcare Information System, so it can be partially compared to other approaches. In order to compute the risk, the Shannon Entropy concept is used as a measure of the uncertainty of a doctor's access to a medical record. By adopting this approach, the doctor's history access is taken into consideration in order to take a decision. Users, are equipped with a certain amount of quotas each predefined period of time. After requesting an access, the corresponding evaluated risk is subtracted from the user's quotas. These quotas represent a threshold for the maximum amount of risk that the system is able to take for each user. In normal conditions doctors should not exceed their quotas; but the idea explained in the paper is that if a doctor rarely asks for more quotas, the system could grant them considering these as exceptional events. However, if a doctor runs out of quotas frequently, he may be over-accessing patients'

informations, and his risk score should grow. This kind of approach is very similar to the credit card approach used in [18] and [55].

A Risk-aware RBAC (Role-Based Access Control) model is defined in [16]. This is an extension of the well-known RBAC model in which the *Role* assigned to users is the key concept in the access control policy. Three risk models are presented in the paper; the authors see the possibility of using these models separately or combined all together in order to create a more complex risk model that supports all features of the three. The first model performs an evaluation of the risk based on the trustworthiness of the user; the second model focuses on user's competence; the third model looks at the appropriateness with which the permission is associated to a specific role. The risk domain is  $[0,1]$  and several sub-intervals are defined inside. An obligation is defined for each sub-interval, working as a mitigation measure. If the computed risk of the request is below the minimum threshold for the intervals, then the request is granted, without obligations. If it is above the maximum threshold for the intervals, then the request is denied. If, instead, it belongs to one of the sub-intervals, the corresponding obligation is enforced.

Very interesting are the future directions that authors foresee for their work: as first thing, they think about developing a context-risk aware access control model by defining a matrix of risk mitigation measures associated with each permission, where each row represents a different context. Secondly, in order to enforce obligations, they would like to adopt a credit card charging approach, similar to the one defined in [18]. A further step, would be to extend this model to include usage control; in this way, the system should adopt a feedback mechanism in order to become responsive to previous access requests. This mitigation measure model, adopted in several approaches, that defines different sub-intervals in the risk domain and it assigns a different obligation for each sub-interval, could be an interesting direction to work on, because it reacts differently and dynamically depending on the type of access we want to perform. Moreover, it gives to the user a concrete way to understand the possible risk and the sensitivity of the action he wants to take; the riskier the action, the stronger the obligation. [17] extends the previous described approach, mainly focusing on how to manage obligations. The basic idea is to introduce a diligence score  $d(u)$  with domain in  $[0,1]$  for each user. Three approaches can be adopted: a strong restriction, a weak restriction and a more conservative approach. According to the strong restriction approach, whenever a user does not fulfil an obligation in time, diligence score is decreased. A risk function of this kind is implemented:  $risk = \min\{1, risk + (1 - d(u))\}$ . However, in some cases, this mechanism could be too restrictive, so a weak restriction approach is defined. The idea is to deny the access, only for those users' requests regarding permission that incurred in obligations remained unfulfilled. A more conservative approach is also defined, by denying access whenever there is a risk and setting initially  $d(u) = 0$ . Then, an obligation for the user is returned and if this obligation is fulfilled, it could guarantee the access in future by reducing the relative risk. It's possible to do this through a reward approach: when an obligation is fulfilled  $d(u) = d(u) + w$  otherwise  $d(u) = d(u) - w$ , where  $w$  is the weight given by the system to the obligation. These approaches are based on the idea that systems usually are not able to force users to fulfil obligations, so the only way to control their behaviour is to set time thresholds, over which evaluate whether obligations were fulfilled or not. A possible way of improving this situation would be managing to have obligations that are immediately verified by the system, thus resulting in a more restrictive policy.

A very interesting Risk-Based Privacy Aware Information Disclosure model is discussed in [7]. As in [76], the paper mainly focuses on confidentiality problems. The architecture described in the paper includes three main modules: a *Risk-Aware Access Control Module*, a *Risk Mitigation Module* and a *Risk Estimation Module*. The access control module includes PEP, PDP and PIP functions. The *Risk Estimation Module* is in charge of computing the risk associated to the request and the *Risk Mitigation Module* decides which type of mitigation measure can be applied in order to reduce the risk. The model is based on the Risk-aware RBAC principles described in [16] but a new kind of mitigation measure is introduced: *Anonymization*, which is the obfuscation, in part or completely, of the data. Let's suppose to have a table of patients; the attributes in the table are

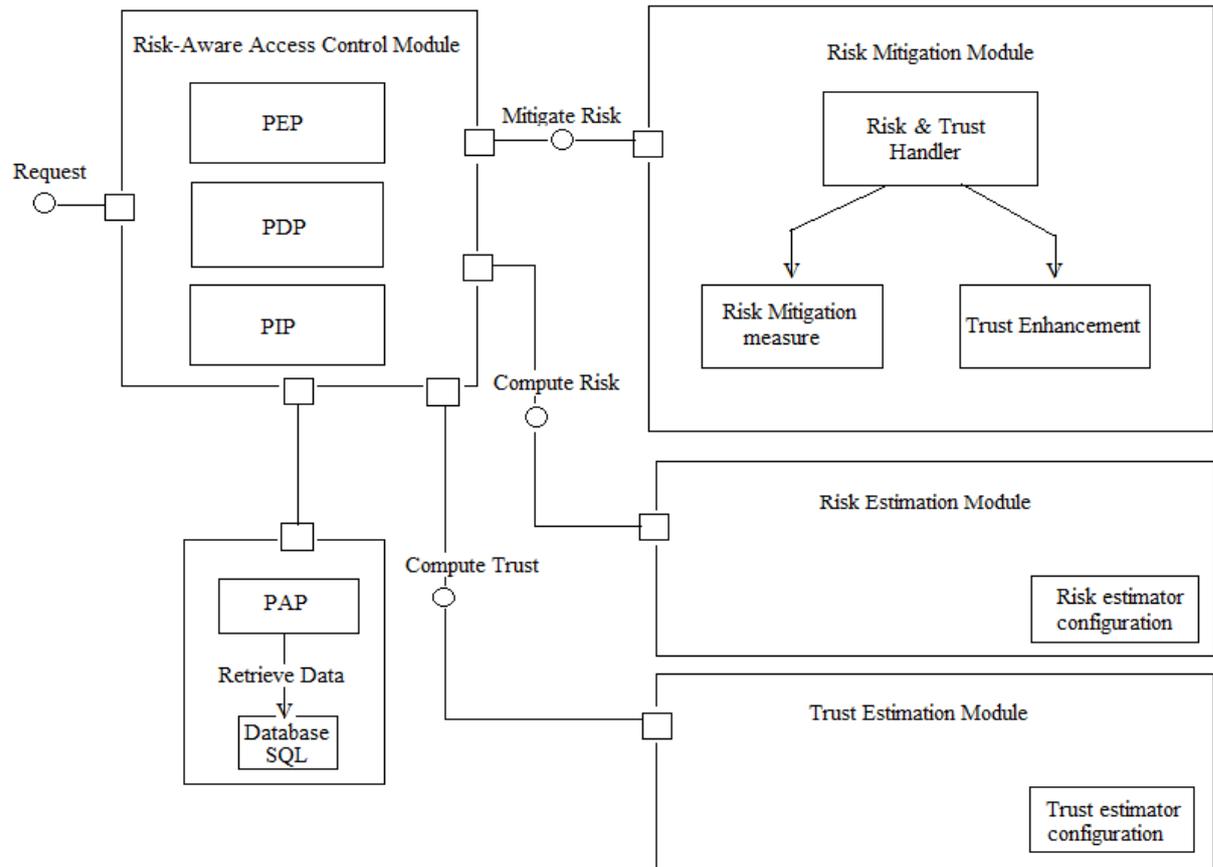


Figure 2.2. A Risk-Based Privacy Aware Information Disclosure model enhanced with Trust computations.

classified as: *Identifiers*, *Quasi-Identifiers*, *Sensitive attributes*. *Identifiers* are data through which an individual can be uniquely identified, *Quasi-Identifiers* are data that if put together can help to identify an individual and finally *Sensitive attributes* represent sensitive informations about individuals. In this model, the concept of trust is limited to be just an attribute and not something to be computed dynamically.

According to the risk computed, a mitigation metric is defined and to each mitigation metric is associated a different view of the information object. We will have  $risk = 1$  if the the user cannot be granted to that permission, otherwise  $risk = 1/p$  where  $p$  denotes the anonymity of a certain view of the table. This approach is better than the one described in [76] because, even if some percentage of risk has been associated to the access, the system, instead of denying the permission, can deliver some obfuscated informations that could eventually be useful.

In [6] we can see an improvement of the architecture described in [7]. A new module is proposed: the *Trust Estimation Module*, which is in charge of computing the trustworthiness associated to the request, a similar concept can be found in [78]. A possible architecture for the framework proposed can be seen in figure 2.2. In this situation, the access is granted if the trust overcomes the risk. Both, trust and risk, are evaluated according to different parameters taken from the context, such as the location of the requester. Whenever the access is denied two options are available: decrease the risk (by enforcing some mitigation measures) or increase the trust level, by fulfilling some obligations. Again *Anonymization* is used as a mitigation measure in order to obfuscate data whenever the associated risk is too high.

In [66], two dynamic risk-based decision methods for access control systems have been proposed. At first, authors propose a new framework, which is an extension of the traditional XACML

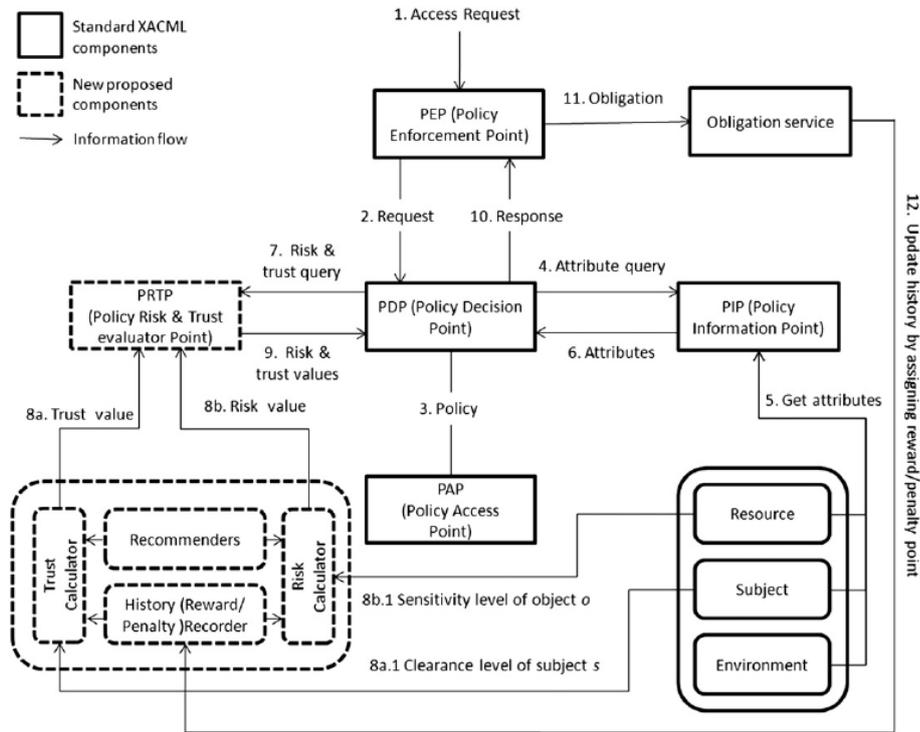


Figure 2.3. High level functioning of the framework proposed in [66].

system. In addition to the PEP, PDP, PIP, PAP we now have a new module called PRTP (Policy Risk & Trust Point). The graphical representation and the information flows of the high level architecture of the framework proposed by the paper are shown in figure 2.3. Whenever an access request is received by the PEP, it forwards it to the PDP, which looks if the organizational policy, deployed on the PAP, forces to evaluate risk and trust for that kind of access, then it fetches all relevant attributes from the PIP. After having done that, the PDP forwards the request to the PRTP module which evaluates risk and trust values based on the past behaviour of the user. The past behaviour is evaluated based on the history of reward and penalty points. The key concept of this framework is that for each user and resource, the access of the user to the resource should be respectively relaxed or restricted, whether the user has shown a positive or negative record of use toward the resource. This is different from RAdAC [50] where history is used to help to fine tune the algorithm in order to adjust the policy and it does not relate with attributes and their value in the system. Moreover, probably, this is a more dynamic and responsive approach than the one used in FuzzyMLS [18] or FuzzyBLP [55], in which we have fixed categories for objects and subjects, that can be only changed manually from an administrator, and no dynamic related to good or bad behaviours of past transactions is present.

After these premises, two adoptable methods are proposed; they both rely on the fact that each subject (user) has a clearance level  $l(s)$  and each object has a sensitivity level  $l(o)$ . According to the first method, a *Local Reward History* and a *Local Penalty History* are defined, then *trust* and *risk* are computed, respectively, as a function of the *Local Reward History* and of the *Local Penalty History*. Finally, if  $trust \geq risk$ , access is granted, otherwise no. According to the second proposed method, instead, history is weighted differently; a weight has to be assigned to

the last access performed and another weight to the previous history. This expedient has to be applied both to reward and penalty histories. Again, if the trust value computed overrides the risk value, access is allowed. One thing to notice is that *trust* and *risk* functions are built in such a way that, in absence of history, it can never happen that a subject, with clearance level smaller than object's sensitivity level, manages to access the object.

## 2.2 Introducing benefit analysis process

BARAC [78] is an attempt to balance risk and benefits that would result from an access. According to this approach, even if the risk computed is high, the access to the resource is allowed if the benefits derived from the access result to be higher. So, actually a risk that in a Fuzzy MLS [18] or Fuzzy BLP [55] would be considered too high to take, here could be overcome by the benefit's factor, changing completely the result of the decision. The BARAC model considers subjects, objects and transactions (read and write). According to the trustworthiness of subjects and sensitivity of objects, an Accessibility Graph and an Allowed Transaction Graph are built in such a way that vertices represents subjects and objects. Transactions corresponds to edges and are associated to risk and benefit vectors. By building a graph, even if the resource is not directly accessible through direct edges, it is possible to find alternative paths between a subject and an object on the graph, where all the transactions are accessible and allowed. In this way, if the benefits outweigh the risks, and the profit ( $profit = benefit - risk$ ) of the Accessibility Graph cannot be further improved, by adding or removing transaction from the graph, then the access to the resource is granted. This model does not give us any hint on how risk and benefit vectors can be computed or how the trustworthiness of subjects can be defined. Moreover, it's for definition a static model relying on a static graph, so actually does not add any dynamic in respect to Fuzzy MLS or Fuzzy BLP, which rely on static categories as well.

In [42] authors propose a risk-aware approach to support UCON (Usage Control) model. UCON is in some way a further evolution of access control models, because it checks the correctness of policies not only during the first access but even afterwards, during usage of resource, being, in this way, a particularly relevant solution for highly dynamic systems. The proposed approach is based on a Markov model, by building a Markov chain, which indicates how the value of an attribute changes. States, in this chain, are possible values of the considered attribute and transitions are possible changes of the attribute. Transition probabilities don't change and are computed using statistics about past operations. When attributes change over time, there can be two possible decisions: continue the session or revoke the session. The idea behind the analysis is to compare the benefits of allowing access and revoking it. When the policy is not any longer met, mitigation measures can be applied, such as the suspension of the usage, until a new fresh and good parameter is received, or letting the session continue and notifying it to a higher authority. The model explained is mainly theoretical and it is more focused on how to map parameters and probabilities in the Markov process. The idea, though, is quite interesting.

## 2.3 Supporting the new Cloud Computing paradigm

It is possible to find some attempts that exploit a Risk-based access control model for the Cloud Computing paradigm, which is one of the most promising and used computing architectures, thanks to its great flexibility, scalability and reduced costs. Still, many improvements can be done toward cloud federations, where the interoperability of different cloud providers becomes one of the main issues. The identification, authentication, authorization of users is for sure one of the

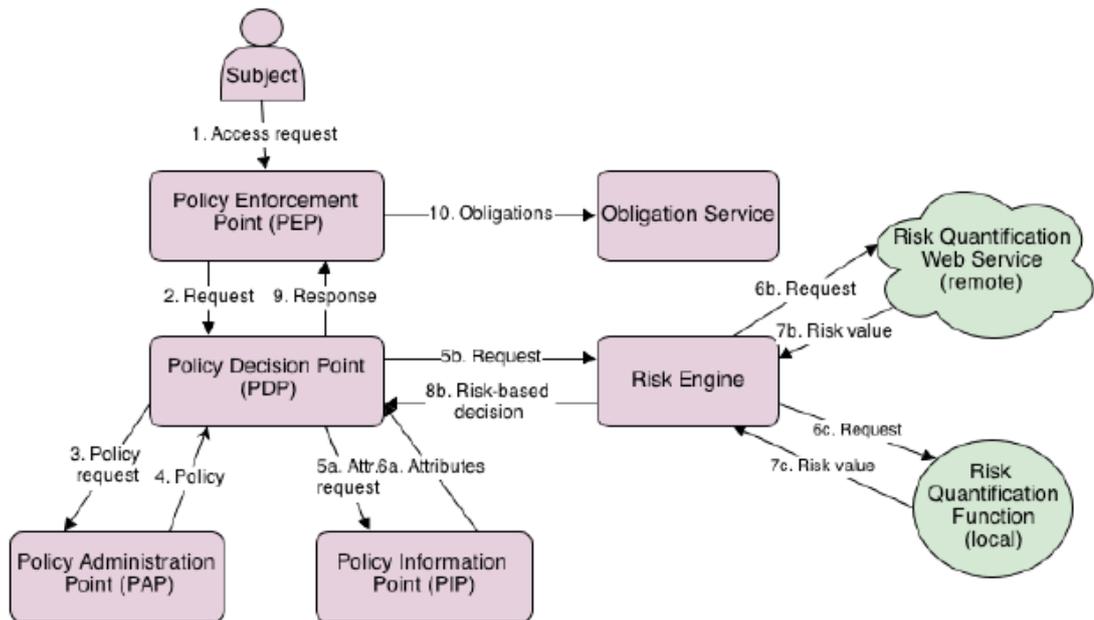


Figure 2.4. Decision process flow of the architecture described in [24].

main topics.

The proposal we find in [24] is an access control model based on an extension of the XACML standard with three new components: the *Risk Engine*, the *Risk Quantification Web Services* and the *Risk Policies*. Whenever an access request is received by the PEP, it is forwarded to the PDP which is in charge of retrieving the traditional XACML policy from the PAP, adding contextual information given by the PIP. In parallel, if a risk policy is associated with the resource, the PDP forwards the request to the *Risk Engine* too. The *Risk Engine* retrieves the risk policy to be adopted for the resource and thanks to a quantification function it computes the risk values for the metrics used and aggregates them. The whole decision process is resumed in figure 2.4. If the risk quantification is done locally, a function in the risk engine itself is executed, if, instead, it is done externally, a web service is invoked for the quantification, degrading a little bit the performance due to the HTTP connections that have to be performed. In this way, the *Risk Engine* generates a response and forwards it to the PDP, which uses a combination method (permit overrides, deny overrides, ABAC precedence, risk precedence) to take the final decision. The response is then sent to the PEP which eventually applies obligations.

This is the first model encountered where the PDP, in order to take a decision, evaluates both the risk-based policy and the traditional XACML policy, meaning that, according to the combination method used, it is possible to obtain either a static access control model or a highly dynamic model, adapting to different companies' requirements. The authors propose to apply this risk model to allow a cloud federation without the need of an Identity federation (which guarantees the trust among users of different cloud providers in the federation). The trust issue will be considered a risk metric in the system. A user can decide to instantiate a resource either in his cloud provider or in a foreign cloud provider, defining a policy for the resource. If someone wants to access a resource belonging to his cloud provider, the request is handled by the home cloud manager which evaluates the request according to the model defined, otherwise the request is forwarded from one cloud manager to the other until the one, responsible of the cloud provider which holds the resource, is reached.

In [67] is presented a prototype implementation of a cloud-assisted eHealth system based on a

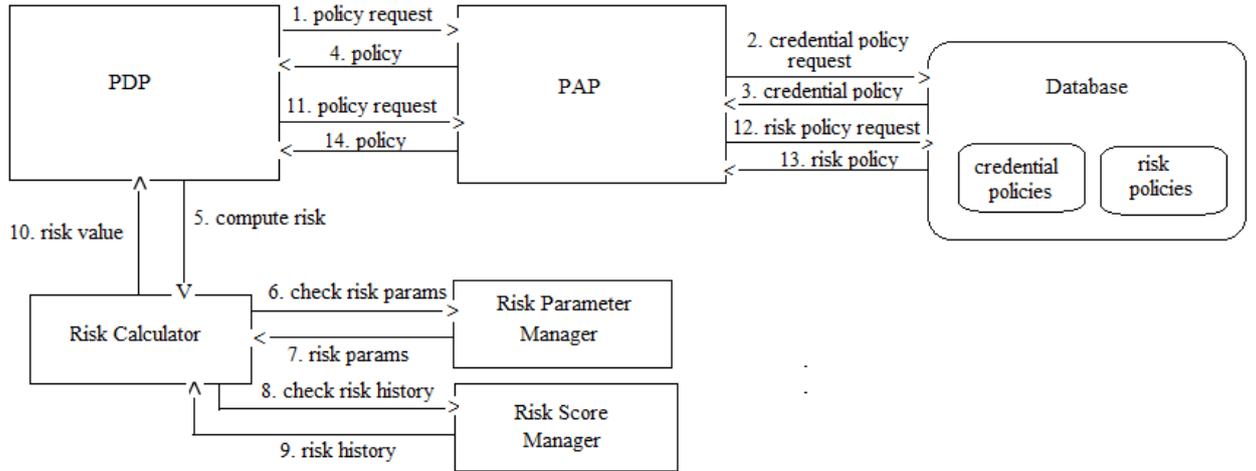


Figure 2.5. A possible information flow derived from the model presented in [67].

risk model to perform access control. The authors actually address the described access control technique as a Task-based AIC where AIC stands for Availability, Integrity and Confidentiality. The access control is performed in two steps: first, a user is checked with credentials; by using credentials, the system identifies the role of the user which is associated to some permissions, like in a RBAC model (which alone is considered to be too static for the cloud scenario). Second, a user is examined with risk. The risk score is computed by looking at the action the user wants to perform, the sensitivity of the resource, the cost in terms of Availability, Integrity and Confidentiality and the previous user's behaviour pattern, like in [66]. Three new components are defined: the *Risk Parameter Manager*, which returns the risk parameter based on the user action, the *Risk Calculator*, which computes the risk score, and the *Risk Score Manager*, which maintains risk score for each user. Figure 2.5 describes the information flow just described. We can say that this model does not add any new concept or idea that wasn't exploited before, but the scenario for which it can find its applicability could be really interesting.

A very basic attempt, instead, to implement Risk-based access control in cloud computing is presented in [45]. The idea is to compute the risk by putting together 8 different risk parameters: years of experience, designation, defect level, referral index, location index, time index, probationary period. For each one of this, a table of reference values exists in order to understand the corresponding risk's value associated. Although no new innovations are brought by this paper, interesting are the parameters taken into consideration in order to compute risk.

In [15] we can find another attempt to describe a Risk-based access control model for Cloud Computing, called DRAC (Dynamic Risk Access Control). The model proposed is based on ABAC (Attribute-based access control). The framework consists of four modules: rule-based detection (RD), risk assessment based on data stream (RA), integrated decision making (ID), and dynamic-threshold setting (DT). An access request consists of two vectors: *req* and *env*, *req* vector is built with some attributes related to the request, *env* vectors, instead, is built with attributes related to the subject (authors suppose there exist some system able to provide this informations). The RD module is the one built according to ABAC principles, so it provides a static way of performing access control by defining attributes, as KEY-VALUE pairs, and policy related to combinations of attributes. The RA module instead is the most important one because enables the dynamic security control. The risk assessment is performed by considering three factors: the instantaneous risk value, the reference metric of the same kind of host and the historical impact of the host itself. These three metrics are computed according to specific formulas and then put together and weighted in order to have a single result in [0,1] domain. The DT module is in

charge of computing the dynamic threshold to compare to the risk assessment. This threshold is computed by taking the history of accesses of the subject and performing the average of the products between the risk associated to each request, the feedback value (0 if it has been discovered to come from a malicious source, 1 otherwise, according to the trust concept) and the sensitivity of the action. Finally, the ID module takes the last decision. If the request has been accepted by the ruler policy, then we proceed with the risk assessment by comparing the threshold with the risk value computed and the sensitivity of the action, otherwise we just rely on the ABAC model implemented in RD module. The idea of this framework is interesting because it improves in some way other similar models previously described: it takes into account the history of past requests in a finer-grained way, by looking both at the history of the host itself and the history of hosts with similar characteristics; moreover, the dynamic behaviour of this model can be put aside, letting the static approach of ABAC to prevail when the situation requires it.

In [32] it's possible to find a further attempt to provide a Risk-adaptive authorization mechanism for Cloud Computing. The model is called RAdAM and it can be adapted to three different Cloud Computing contexts: Cloud User, Cloud Collaborations, Cloud Federations (which are collaborations on a wider scale). Each request is characterized by an entry  $\langle Subject, Object \rangle$  associated to a risk value. A threshold is also defined and attached to the subject and the object. Moreover an average risk value and an average threshold are defined for a user, even though, no clues on how to compute these risks are given by the authors. In the collaboration scenario we also have a collaboration access request's risk and in the federation scenario we have a federation access request's risk. The architecture is presented as an extension of the traditional XACML standard with the introduction of the RAdAM module, which is called by the PDP when an access request is performed in order to compute the risk, see figure 2.6 for a more complete representation of the modules and information exchanged proposed. Authors propose also a way of implementing the RAdAM model with *fuzzy* logic. They decided to represent the clearance of the user in a trapezoidal membership function and the object sensitivity in a combination of trapezoidal and triangular membership functions. The process is very similar to the one used in [55]. An enforcement of the RAdAM model with the Vulnerability Based Authorisation Mechanism is also proposed which is exactly equal to the one proposed in [18] where sensitivity labels are assigned to objects and subjects, where a subject can access an object only if its label dominates the object's label. This proposed model, anyway, is quite abstract and not much details are given to explain the risk's computation process.

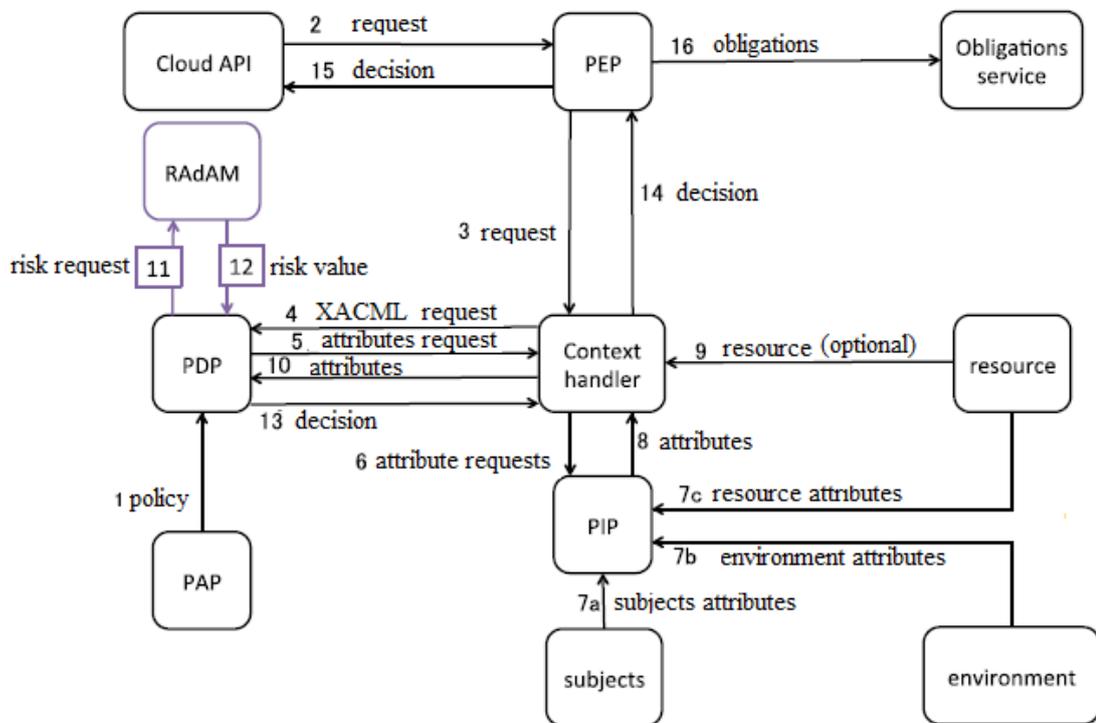


Figure 2.6. XACML-RAdAM architecture proposal, taken from [32].

## Chapter 3

# Requirements for a Risk-Based Access Control Framework

After having understood, from the state of the art analysis, which are the currently proposed solutions for the implementation of a risk-based access control framework, we try, in this chapter, to extract the main concepts observed, in such a way to better understand the necessary requirements to build an access control solution of this kind. To this purpose, a model, that resumes all the main features observed in the currently proposed solutions supporting a risk-based decision logic, will be presented. Then, this model will be extended in order to integrate, in the final solution, those secondary requirements needed to further develop the previous model, by slightly changing the way in which access control decisions are taken and are enforced by the system. Finally, security implications coming from these models will be outlined.

### 3.1 Architectural Requirement Analysis

#### 3.1.1 Risk-Based Approach

The state of the art analysis, performed in Chapter 2, highlighted how currently proposed solutions do not differ significantly when it comes to propose an architectural model over which developing a risk-based access control solution. Some discrepancies emerge, instead, in the way risks should be accounted and calculated, showing how the concept of risk could be somehow relative and often related to the context in which it is considered, although some general definitions can be addressed to it.

Since the purpose of this thesis work, is not to define a methodology for computing risk, but it is to develop an access control framework able to exploit risk when taking authorization decisions, relying on an external provider in charge of managing and storing those risks, we are going to focus, on the architectural requirements of this framework.

The most evident requisite, that emerges from the proposals, is the necessity of developing a solution relying on an easily extensible and scalable approach, which is generally achievable through the adoption of the eXtensible Access Control Markup Language (XACML) standard architecture.

XACML is an OASIS standard [72], based on XML syntax, and has become the *de facto* standard for policy specification and evaluation due to its great success and large use among applications and platforms, as observed in [46]. Since basically all the mentioned approaches, which provided an architectural model to support their proposals, based their solutions on the standard XACML architecture, it is worth to briefly mention all the components involved in this architecture (see figure 3.1):

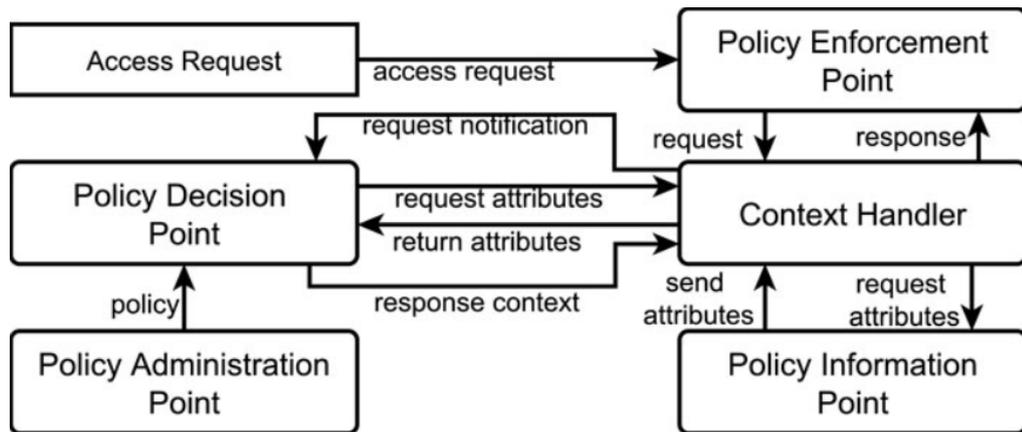


Figure 3.1. XACML architecture overview (taken from [40]).

- the Policy Enforcement Point (PEP) is in charge of protecting a resource, by handling the access request, and, only after a proper verification of compatibility with the policy has been performed, granting the access to the resource;
- the Policy Decision Point (PDP) is the entity which receives all the data, linked to the access request, and the policy to apply, and decides whether to allow or not the access;
- the Policy Administration Point (PAP) is the entity which provides the policy applicable to the submitted request;
- the Policy Information Point (PIP) is the component which provides additional informations related to the access request, the subject and the resource;
- the Context Handler (CH) is the component which translates access requests (and responses) to (from) a valid XACML format, if they are still not, moreover, it enhances the requests with attributes (taken from the PIP) related to the context in which the requests are performed.

Let's suppose a user wants to access a document, present on the intranet of the organization for which he works, and the policy adopted by the access control system has been configured to discriminate access requests by looking for the email address associated to users. When the access request, in which, in such a situation, the subject is identified by his username, is submitted from the application, it is handled by the PEP, which is generally tightly bound to the application. The PEP temporarily block the access request and, since it does not posses the necessary informations to evaluate itself the request, forwards it. The application request, which is not necessarily already in XACML format, is received by the CH, which converts it in XACML and forwards it to the PDP. The PDP retrieves the policy, to be used for the evaluation of the request, from the PAP, which takes the requested policy from the policies repository where they are deployed. If additional informations, like the email address associated to the subject, are needed in order to correctly evaluate the request, the PDP asks again to the CH to provide these informations, that the CH retrieves from the PIP. At this point, the PDP, by using all the data in its possess, evaluates the access request and formulates an XACML response which is returned to the CH. The CH, if needed, translates the XACML response in the application format and send it to the PEP, which enforces the decision taken and communicates it to the user. The final decision can be either permit or deny or indeterminate, if no sufficient informations were present to correctly evaluate the request, or not applicable, if there was no policy to apply to the submitted request. After having defined the core architecture supporting access control decisions, arise the necessity

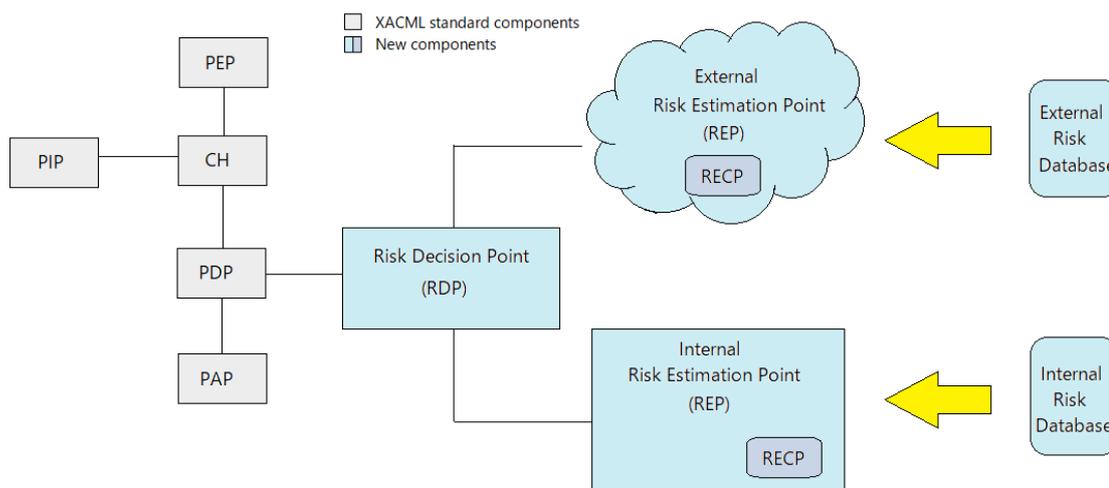


Figure 3.2. The proposed Risk-Based Access Control basic architecture.

of having specific components that allows to manage risk. A risk engine is needed, able to compute the risk values and to evaluate them according to some defined rules. The two operations are quite distinct and, in general, there is no predefined constraint in considering the entity responsible to compute the values of risk specifically as an internal component of this risk engine, indeed it can be easily outsourced, as the work presented in [24] clearly shows. For these reasons, it would be clearer to separate them in two diverse modules. We can, thus, identify a Risk Decision Point (RDP) as the module responsible of evaluating the risk according to a specific implemented mechanism, thus implementing all the functions and methods characterizing the core logic adopted when taking the decision based on risk. The risk will be retrieved from a Risk Estimation Point (REP), a module responsible to compute the risk values for the specific access request. Since, as aforementioned, this module can be either an internal component of the access control framework or an external entity appointed to do the work, we talk about Internal Risk Estimation Point and External Risk Estimation Point. The proposed architecture is shown in figure 3.2.

As it is possible to see from the picture, internally to the Risk Estimation Point, another component can be identified: a Risk Estimator Configuration Point (RECP), which represents a necessary interface that allows to manage and configure the risk estimation functions adopted when quantifying the risk values.

Furthermore, a relational database to store risk values is often needed, either to maintain the risk score of resources or, in some cases, to store values referring to user's risk history, either directly, like several approaches suggest ([50], [76], [66], [42], [67], [15]), or indirectly ([18], [55]) by considering access quotas or credits.

In this way, the basic requirements, emerged from the state of the art analysis, necessary to implement an access control framework, which use just the concept of risk in order to take the final authorization decision, have been presented and outlined. They can be resumed as:

- **Req.1:** Implementing an XACML-based Access Control framework.
  - **Req.1.1:** Implementing PEP component.
  - **Req.1.2:** Implementing PDP component.
  - **Req.1.3:** Implementing PIP component.
  - **Req.1.4:** Implementing PAP component.
  - **Req.1.5:** Implementing CH component.

- **Req.2:** Relying on a risk estimation module (Risk Estimation Point).
  - **Req.2.1:** Implementing a database able to store risk informations.
  - **Req.2.2:** Connecting the database to the network in order to be able to retrieve risk scores.
  - **Req.2.3:** Defining the necessary parameters to be used in the risk computations.
  - **Req.2.4:** Defining the necessary functions to use for the final risk computations.
  - **Req.2.5:** Implementing an interface from which associating risk parameters to resources.
  - **Req.2.6:** Implementing the necessary interfaces to receive and manage risk requests.
  - **Req.2.7:** Installing the module on a server, connected to the internet or to the internal network (eventually, if the service has not been outsourced), to let the service be contacted, in order to retrieve risk informations related to an access request.
  
- **Req.3:** Implementing a risk based decision module (Risk Decision Point).
  - **Req.3.1:** Implementing the necessary interfaces to build a risk request, in a format supported by the Risk Estimation Point.
  - **Req.3.2:** Implementing functions supporting the risk decision logic.
  - **Req.3.3:** Implementing an interface from which the module can be contacted by the PDP.
  - **Req.3.4:** Installing the module on a server, connected to the internet, in order to be able to contact the Risk Estimation Point (if outsourced), and to the internal network, in such a way to let the service be contacted by the PDP, during the evaluation process of an access request.

### 3.1.2 Trust & Risk-Based Approach

After having outlined the basic requirements strictly needed to successfully implement a risk-based access control framework, we can introduce a development to the proposed model, trying to understand, even here, the requirements necessary to support this proposal, always following the guidelines provided by the analysis of the state of the art concerning systems of this kind.

One of the mostly enumerated and prominent proposals, we can find, is related to the use of the trust concept. Trust can be a relevant value when considered within an authorization decision, indeed it represents the degree of confidence the system puts in a subject requesting the access towards a resource.

Similarly to risk, also trust needs to be computed according to some predefined parameters and thus explicitly quantified. For this reason, it would be appropriate to have a separate module, as for risk, which deals with the trust estimation problem. Works like [6], [66] support this idea. We are going to address this module as the Trust Estimation Point (TEP), that again can be either external or internal, depending whether the system foresees to outsource the service or not. Within such component, a Trust Estimation Configuration Point (TECP) should be present, as an interface to configure trust parameters and functions. In such a situation, moreover, a database enhanced to provide, beside risk, also trust values would be necessary. In figure 3.3 the proposed architecture, able to enhance the risk-based paradigm with trust computations, is proposed.

As it is possible to see from the picture, now, RDP has become a Risk & Trust Decision Point (RTDP) where both risk and trust values are considered when taking the authorization decision. Although the integration of the trust related modules can be considered an interesting direction to take in order to implement a dynamic access control mechanism, the requirements needed to

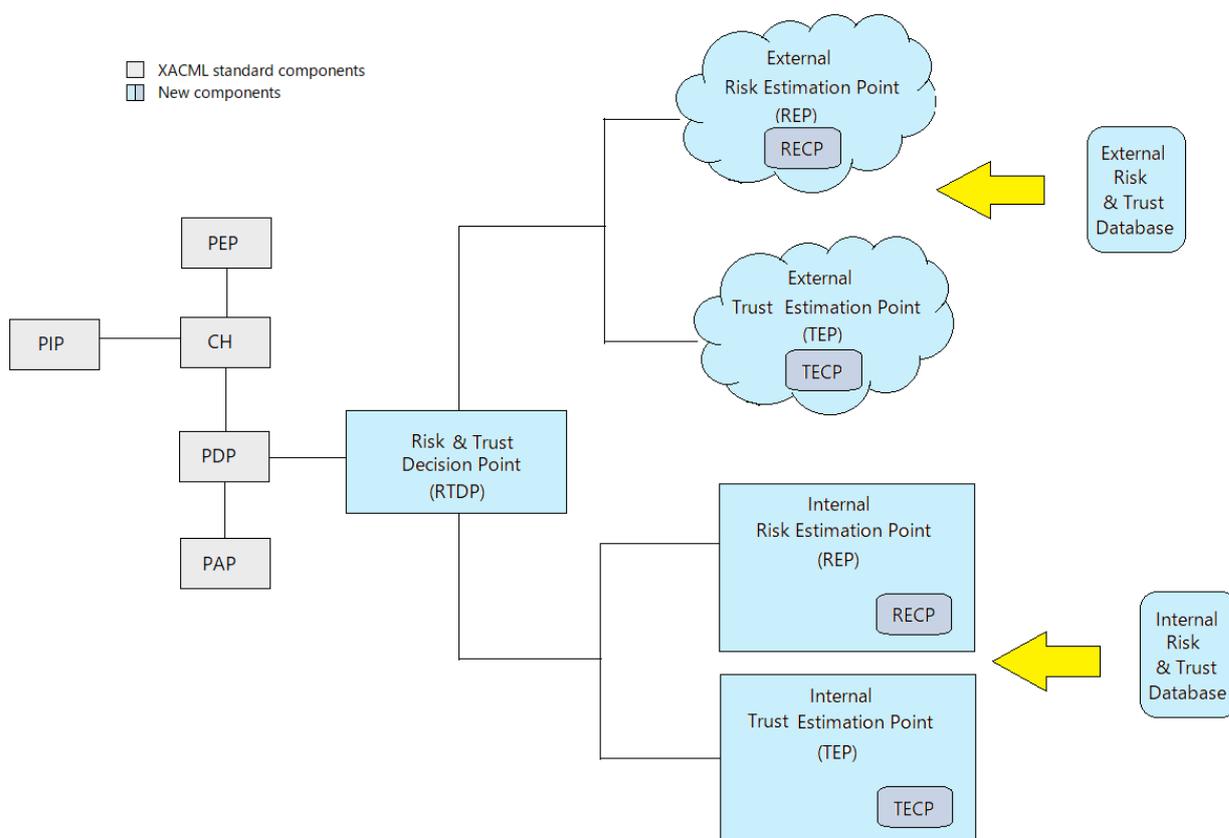


Figure 3.3. The Risk-Based Access Control architecture enhanced with the Trust Estimation Point.

support such a solution cannot be considered with the same weight of the ones identified before, since they just support a further evolution of the already presented system, adding to it a new kind of functionality, related to the way the authorization decision is taken. For this reason, they are going to be addressed as additional requirements and they can be resumed as:

- **Add. Req.4:** Relying on a trust estimation module (Trust Estimation Point).
  - **Add. Req.4.1:** Implementing a database able to store trust informations.
  - **Add. Req.4.2:** Connecting the database to the network in order to be able to retrieve trust scores.
  - **Add. Req.4.3:** Defining the necessary parameters to be used in the trust computations.
  - **Add. Req.4.4:** Defining the necessary functions to use for the final trust computations.
  - **Add. Req.4.5:** Implementing an interface from which associate trust parameters to resources.
  - **Add. Req.4.6:** Implementing the necessary interfaces to receive and manage trust requests.
  - **Add. Req.4.7:** Installing the module on a server, connected to the internet or to the internal network (eventually, if the service has not been outsourced), to let the service be contacted, in order to retrieve trust informations related to an access request.
- **Add. Req.5:** Implementing a risk & trust based decision module (Risk & Trust Decision Point).

- **Add. Req.5.1:** Implementing the necessary interfaces to build risk requests and trust request, in a format supported by the Risk Estimation Point and Trust Estimation Point.
- **Add. Req.5.2:** Implementing functions supporting the risk & trust decision logic.
- **Add. Req.5.3:** Implementing an interface from which the module can be contacted by the PDP.
- **Add. Req.5.4:** Installing the module on a server, connected to the internet, in order to be able to contact the Risk Estimation Point and the Trust Estimation Point (if outsourced), and to the internal network, in such a way to let the service be contacted by the PDP, during the evaluation process of an access request.

### 3.1.3 Obligations Enforcement

The last major concept it is possible to identify, in the analysed proposals, is obligation enforcement. The purpose of obligations is to mitigate, in some way, the risk that the system is willing to take when granting an authorization. There are diverse methods, it is possible to find in literature, to adopt obligations, [18] and [55] propose, for example, a model where obligation enforcement should be adopted immediately, by the system, when the access is permitted, by decreasing the risk credit owned by the user. Authors, in [17], instead, argue that existing systems are not able to guarantee the enforceability of obligations that have to be fulfilled by users, thus not automatically by the system. To this purpose, they propose a model where obligations can be fulfilled in a larger time interval and thus controlled by the system.

The XACML component, which is generally involved in the enforcement of obligations is the PEP, which should either enforce itself the obligation or rely on an separate obligation enforcement service. The correspondent risk and (or) trust values, that have been retrieved, should be communicated from the PEP to the obligation service, in such a way it can choose what kind of mitigation measure to adopt. This means that the Risk & Trust Decision Point should include in the response, not only the decision taken, but also the risk and (or) trust values, that are communicated to the PDP, which should include them in the XACML response returned to the PEP. At this point the obligation service either will communicate itself to the user the obligation needed to fulfil in order for the access to be granted, or it will pass again the information to the PEP, that will enforce the obligation. Figure 3.4 shows the access control architecture enhanced with the obligation module. We can thus identify the following additional requirement:

- **Add. Req.6:** Implementing an obligation mechanism.
  - **Add. Req.6.1:** Defining the set of supported obligations.
  - **Add. Req.6.2:** Implementing a mechanism to bind risk and (or) trust scores to the defined obligations.
  - **Add. Req.6.3:** Implementing the necessary interfaces to make the PEP and the Obligation Service to communicate with each other (if implemented separately from the PEP).
  - **Add. Req.6.4:** Installing the module on a server, connected to the internal network, in such a way to let the service be contacted by the PEP (if implemented separately from the PEP).

## 3.2 Security Implications

As previously stated, there can be two adoptable approaches when estimating risks: the organization, which implements the access control mechanism, can decide either to outsource the service

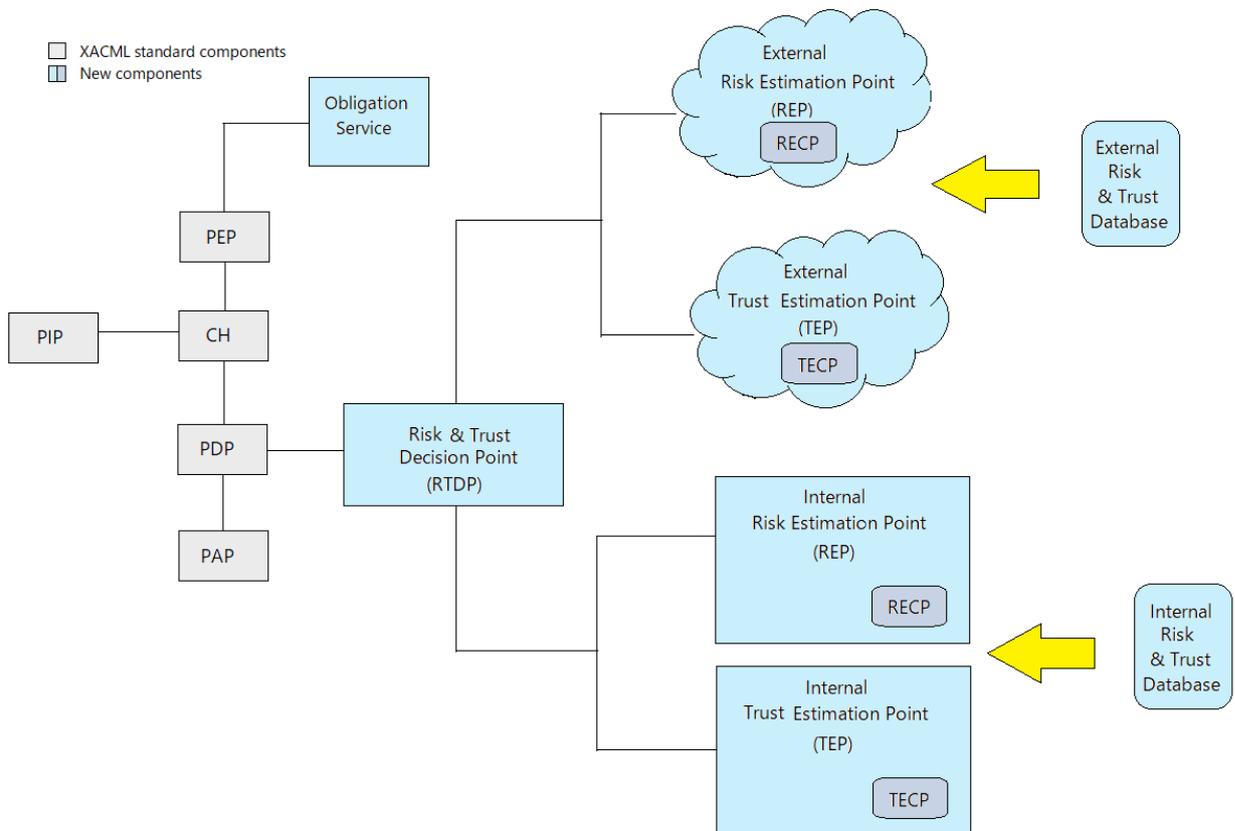


Figure 3.4. The Risk-Based Access Control architecture enhanced with the Trust Estimation Point and obligation enforcement.

to an external trusted entity or to implement it internally. By letting the possibility, to our system, to rely on external providers in order to retrieve informations supporting access decisions, we enhance the extensibility of our framework, that can, in this way, exploit additional informations coming from trusted sources. On the other end, some security implications arise, in the communication flow between the organization's access control system and the risk's provider, that have to be taken in consideration in order for the whole system to work properly and in a secure way. If we observe the developed risk & trust based access control architecture, there can be at least three categories of attack it is possible to identify, as figure 3.5 shows: Man In The Middle (MITM) attacks, Denial of Service (DoS) attacks and DNS Poisoning attacks. A MITM attack consist in a malicious third party which inserts itself in a communication, which is happening from two other parties, and manipulate the exchanged information flow. In our specific scenario, the attacker could decide to modify, filter or just read the intercepted data, moreover it could think about storing a copy of the data too. In case of data modification or filtering, the integrity of the data would be compromised. An attack of this kind could involve, for example, an attacker who, for some reasons, wants to fool the access control system by providing it with a smaller risk value for a specific access request, in such a way to grant himself, or to someone else, the authorization over a resource; on the other end, it could also involve an attacker who wants to block some user access, by providing to the system an higher risk value for the resource, or by filtering the response, mining not only the integrity of the data, but the availability of the resource itself too. Another attack, that could be brought by a malicious MITM, is packet sniffing, which consists in a passive behaviour of the attacker, who limits himself to observe the traffic exchanged by the two entities, endangering, in this way, the confidentiality of the data. In such scenario, the attacker could be an untrusted third party that wants to obtain sensitive informations, like credentials,

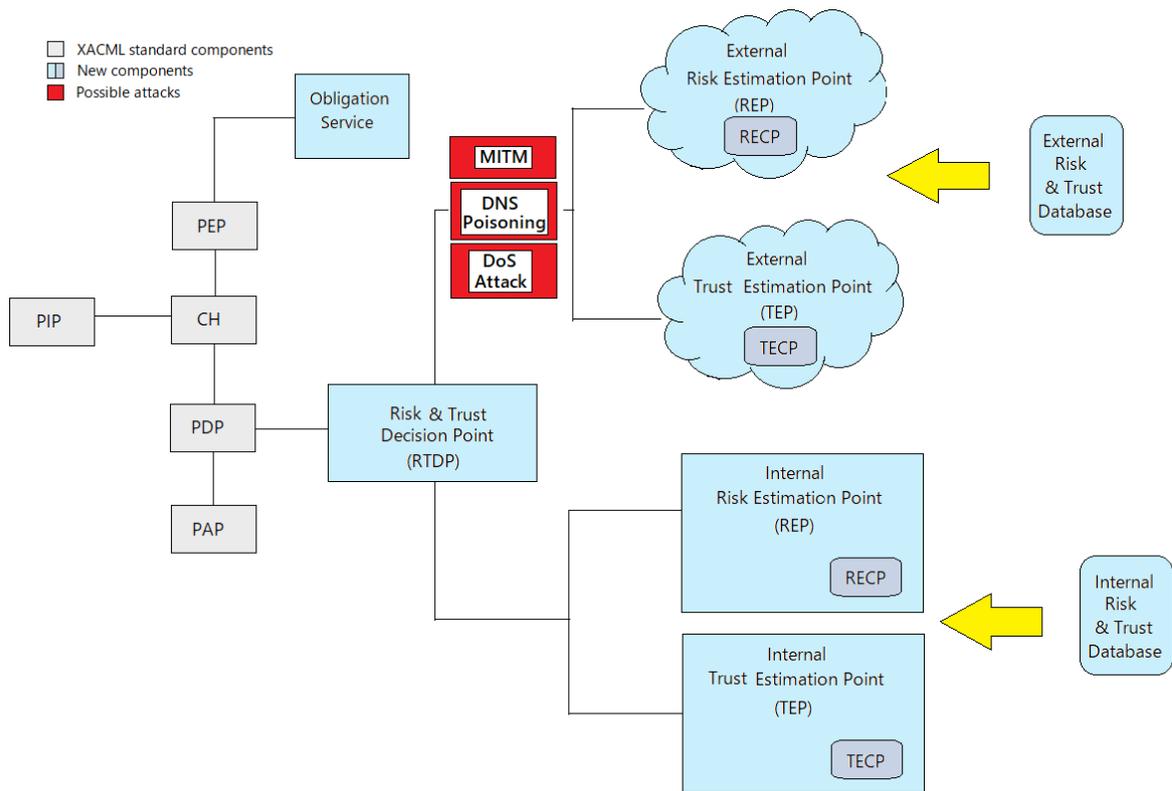


Figure 3.5. Identification of possible attacks endangering the access control architecture which outsource risk and trust estimation.

during the authentication phase of the two systems, or analyse requests and responses, in order to understand the risk values associated to resources and access requests, selling, in a second moment, those informations or keeping them for future purposes.

Finally, the last attack we can address to a MITM is the replay attack. According to this kind of attack, the malicious source intercepts the data exchanged and stores a copy, in such a way to reuse it in a second moment. In this way, an attacker that, for example, makes a copy of a risk score resulting from a certain access request, could resend the same response to other future requests, providing in this way wrong risk values.

The second category of possible attacks, from which an architecture of this kind could be susceptible, is DoS. A DoS attacks aims to make unavailable a service provided by a system, typically saturating its resources. In our scenario, the attack could be directed towards either the External REP or the TEP or the Risk & Trust database, from which the two entities rely to store risk informations. The unavailability of, at least, one of these services makes impossible the functioning and the correct evaluation of the access control decision. A DoS attack can be obtained, for example, by flooding with a huge number of requests the targeted system, overloading it, and, in this way, preventing other legitimate requests to be successfully fulfilled.

Finally, serious issues may arise when considering risks deriving from a DNS Poisoning attack. According to this kind of attack, the DNS cache of a system is maliciously altered with the purpose of attracting the victim towards a server, different from the original one, generally called shadow server, which can provide fake answers. In our kind of scenario, an attacker could execute this attack, for example, to steal authentication credentials exchanged by the two systems, endangering the confidentiality of those informations, or, by providing fake risk scores, to block some user access, mining the availability of the system, or to grant some unauthorized one.

From the identification of those possible attacks, that could represent serious threats for the system,

it emerges, at first, the absolute necessity for the adoption of an authentication mechanism, able to prevent not legitimate users from accessing the risk (or risk & trust) estimation service. Clearly, if the trust estimation service is present and is provided by another third party, different from the one providing the risk service, a second authentication mechanism would be needed with this third party, probably degrading, in this way, the performances of the whole authorization service. At the same time, though, in order for the authorization system to avoid of incurring in possible DNS Poisoning attacks, which would make the system unwillingly talk with a malicious entity, the support of techniques like DNSsec, providing authentication, integrity and non repudiation of DNS responses, should be considered. Anyway, a drawback, that a solution of this kind would bring, is a large overhead, due to the computations of signatures.

Mechanisms, able to provide authenticity and integrity of data, in order to counteract unauthorized manipulations, should be present too. At the same time, data confidentiality should be provided, both during the authentication phase and during the risk (& trust) communication phase, in order to avoid that sensitive informations, like credentials or risk values, could be captured by an untrusted entity. In order to cope with these problems, advanced techniques for encryption and digital signature should be adopted, always considering the drawbacks related to the performance implications that those techniques bring. An anti-replay mechanism would be probably needed too, able to link every response to the corresponding request.

Finally, in order to deal with attacks of type DoS, traffic monitoring techniques, like the adoption of Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), plus some some servers' scalability, would be suggested within the organizational context of the Risk (& Trust) Estimation Point.

## Chapter 4

# Analysis of the Tools: CAIRIS and SAFAX

### 4.1 CAIRIS

#### 4.1.1 Introduction

CAIRIS [12], which stands for Computer Aided Integration of Requirements and Information Security, has been designed as a security requirements management tool with the purpose of integrating security, usability and requirements concepts supporting a system's design process. Before proceeding with the analysis of the tool, it can be useful to explain shortly what these concepts are and why are so important in systems' design. As systems become more complex the probability of introducing security vulnerabilities increases, therefore, in order to provide assurances about the designed system and to avoid that unintentional vulnerabilities are introduced, we need some way of analysing, understanding and managing all the data and the informations that come up from the process. This procedure has to be carried on preferably since when eliciting and specifying requirements, so already at the earliest stages of the design process.

Although works on Security Requirements Engineering exist, it seems, from the literature, no relevant attempts on usability models have been proposed with the purpose of enhancing security models.

In [39] authors depict the usability and security threat model as a particularly critical model when considering legitimate users, which they have no intent in harming the system. A bad designed system, which does not respond to good usability properties for its users, can lead to intentional or unintentional misbehaviours of the users themselves, which, generally, have not the perception of the possible threats they could cause. Similarly, Faily and Fléchais claim in [29] that the integration of security and usability in systems' design needs to be done in order to predict security implications of usability decisions and, vice versa, usability implications of security design decisions.

A standard definition for usability is given by ISO 9241-11 [36]: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Effectiveness is further specified as the "accuracy and completeness with which users achieve specified goals". Efficiency is defined as the amount of "resources expended in relation to the accuracy and completeness with which users achieve goals". Satisfaction as the "freedom from discomfort, and positive attitudes towards the use of the product". Finally, the context of use is defined as the "users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used". Context of use is one of the key concepts over which CAIRIS has been built. In particular,

this concept is expressed by the term *Environment*. An Environment might represent a system operating at a particular time of day, or in a particular physical location. Some concepts are strictly specific of the environment: some of them are more concrete, such as assets, tasks, personas, attackers, and some other, like goals, vulnerabilities, threats, risks, are less. Although the environment encapsulates the majority of the key concepts in security and usability evaluations, during the design process, there exist other factors, like *Role*, for example, which are considered environment-independent. This make sense since the role of an individual is assumed to be something that does not vary by environment, what can vary are the security implications that a role has, while, for example, the goal is something that can significantly change in different environments; same thing for the *Risk*, which exists only if the threat and the vulnerability, that provoke it, are present at the same time in the environment. See figure 4.1 to see a more complete view of the concepts that in CAIRIS have been associated to the environment.

As aforementioned, the last fundamental concept, over which the CAIRIS logic stands, is *Requirement*. The IEEE definitions of Requirement [20] are:

1. a condition or capability needed by a user to solve a problem or achieve an objective;
2. a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents;
3. a documented representation of a condition or capability as in (1) or (2).

Requirements represent, in CAIRIS, the needs and the properties that have to be present or satisfied in a certain context; so they represent the basics of this tool and they are not just security and technical ones but they have to be defined also from the usability point of view. In [26] Faily claims that requirements are typically classified according to whether they are Functional or Non-Functional. Functional requirements define those aspects of conditions that have to be satisfied in order to guarantee that the system will work properly, while non-functional requirements define the *how well* aspects, so they represent quality requirements. Often, security requirements are considered as non-functional ones. From this perspective it could be dangerous to consider security just as an additional aspect to improve systems' design, since it could take developers and stakeholders to think about security only when the system is already mature, letting functional requirements to take precedence.

#### 4.1.2 Baseline Architecture and Goals

CAIRIS tool has been built upon the IRIS (Integrating Requirements and Information Security) meta-model, a conceptual model for Usable Secure Requirements Engineering [27]. The two main innovations that it introduces in the literature of security requirements models are: first, usability of tasks is taken into consideration in a qualitative and quantitative way, exactly as the risks; second, the meta-model explicitly defines concepts and associations which allow a context of use to be modelled according to the specific needs. It is worth highlighting that the usability concept is not just a characteristic of tasks but it is specific for the pair task-persona. Indeed, the effort required by a task is user's specific since it depends on the competence, the abilities and the goals of a user. Similarly, also the degree of how much a task interferes with a persona's goals is something which is strictly related to the individual.

*Personas* are descriptive models of how users behave, how they think and their goals. Personas are created by involving, in the design process, stakeholders who can provide empirical data relative to several contexts of use. The data gathered from the observations and interviews are mapped to behavioural variables, as explained by some already existing works that has been done about how integrating software requirements process with personas, like in [13]; the set of all the

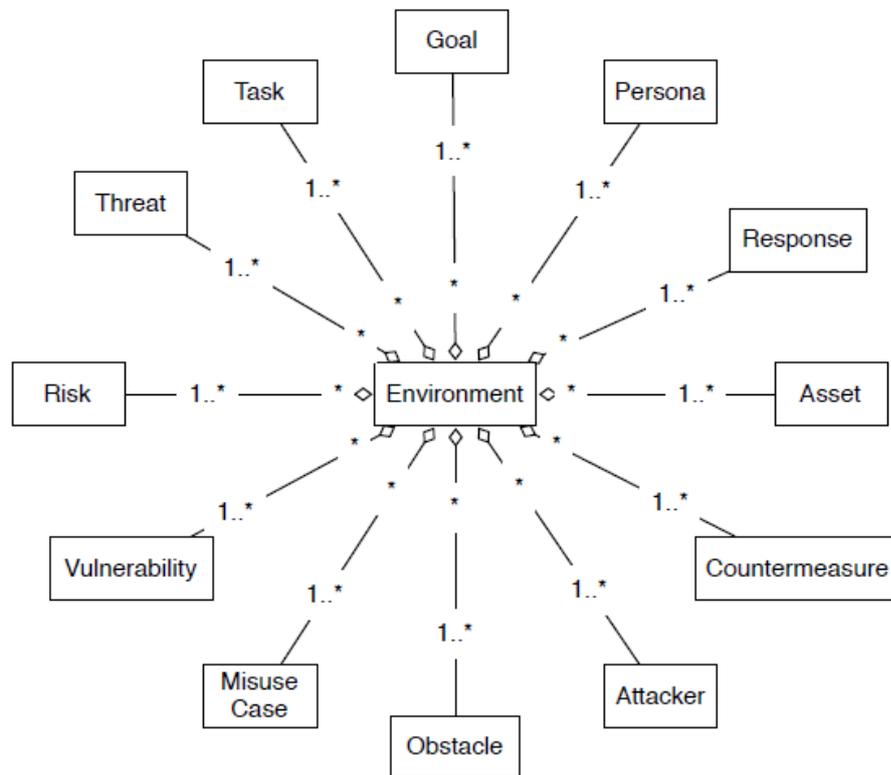


Figure 4.1. The environment meta-model, supported by the IRIS framework, over which CAIRIS has been built [27].

interviewed subjects, grouped within a set of behavioural variables, forms a behavioural model, which constitutes the basis of a persona. This persona-oriented approach can be really useful to create real world scenarios which can help to identify those security vulnerabilities that could arise in presence of attackers or even simply unintentional misuse cases.

An important aspect to consider is that IRIS framework distinguishes between personas (diligent human assets) and attackers (human assets which are harmful for the system), although for both the categories the persona-oriented approach is used to create human profiles. This distinction comes from the fact that, in IRIS, tasks are associated to the persona but responsibilities are associated to Roles; by doing in this way, personas and attackers can share the same role, allowing to make deeper considerations about how eventual inside attackers could create damage.

CAIRIS allows to switch among different models by changing the perspective through which the system can be seen, in other words it allows to analyse requirements from different points of view, specified in the IRIS framework; in particular it allows to build a Task Meta-Model, a Goal Meta-Model, a Risk-Meta-Model, a Responsibility Meta-Model and an Environment Meta-Model [27].

Furthermore, the IRIS meta-model presents a way of qualitatively rating and quantitatively scoring risks and tasks. In [28] are presented the Risk and Task Analysis performed by IRIS.

For what it concerns the Risk Analysis, from a qualitative point of view, two parameters are defined: *Likelihood* and *Severity*, that combined together, according to the likelihood and severity tables in IEC 61508 [19], determine a Risk rating in the range [1, 4] where, starting from the bottom, values correspond to intolerable, undesirable, tolerable and negligible. The value of Likelihood and Severity parameters are then mitigated by the corresponding values for the applied countermeasures according to the formulas:

$$L_r = L_t - \bar{m}_t$$

$$S_r = S_v - \bar{m}_s$$

where  $L_r$  is the likelihood of the threat,  $L_t$  is the likelihood of the threat  $t$  associated with risk  $r$  and  $\bar{m}_t$  is the mean likelihood value for the set of countermeasures mitigating  $L_t$ , while  $S_r$  is the severity of the vulnerability exposed by risk  $r$ ,  $S_v$  is the severity of the vulnerability  $v$  associated with risk  $r$  and  $\bar{m}_s$  is the mean severity value for the set of countermeasures mitigating  $S_v$ .

From a quantitative point of view, in order to score the risk related to specific assets that can be threatened, a row vector  $[c \ i \ a \ o]$ , which represents the security properties associated with an asset, is introduced, where  $c$ ,  $i$ ,  $a$ ,  $o$  represent risk values for confidentiality, integrity, availability and accountability respectively. A new parameter is then defined in order to represent the risk *Impact*, according to the formula:

$$P_r = (P_t \times P_a) - \bar{m}_p$$

where  $P_r$  is the risk impact, represented by the security properties held in the assets at risk from risk  $r$ ,  $P_t$  represents the security properties of the threat associated with risk  $r$ ,  $P_a$  represents the security properties of the vulnerable or threatened assets at risk and  $\bar{m}_p$  represents the mean security properties for the countermeasures targeting the risk's threat or vulnerability.

The final risk is computed as:

$$Risk = L_r \times S_r \times P_r$$

where  $L_r$  represents the likelihood of the threat,  $S_r$  represents the severity of the vulnerability exposed by the risk and  $P_r$  represents the risk impact in terms of security properties endangered. In this way, [Req.2.3](#) and [Req.2.4](#) have been addressed.

As Faily and Fléchaïs observe in [27], there can be different ways of treating risks: they can be accepted “if we are prepared to accept the consequences of their impact”, ignored “if the responsibility for dealing with them is out of scope”, or mitigated “if treating the risk has a bearing on the system specification”, thus the vulnerabilities, which originate the risk, are an intrinsic part of the system and the risk cannot be neglected. In the same paper, authors define the relation between risk management and system design by saying that “choosing to mitigate a response is synonymous to intentionally specifying that the system shall manage the risk as part of its design”, and this is what Security Requirements tools and, in particular, CAIRIS allow to do. When mitigation's countermeasures are applied, the value of each one of the parameters aforementioned is reduced according to the effect that the countermeasures will have on that risk parameter.

Choosing a certain countermeasure, in order to reduce the risk, is not enough though, indeed, the mitigation of a risk could result in a worse task's usability for a user; this is why IRIS presents also a Task Analysis [28], in which usability parameters are involved. Since usability is something strictly related to the persona, there is not a predefined association between usability parameters and a task, but, more precisely, usability parameters are associated to a task carried out by a certain user. Four properties are set for each persona participating in a scenario: *Duration*, *Frequency*, *Demands*, *Goal conflict*. Each one of these properties is mapped to usability components defined in ISO 9241-11 [36]. In particular, Duration and Frequency of the task refers to efficiency, while Demands (mental and physical) refers to satisfaction and Goal conflict refers to effectiveness, which is the degree to which the task interferes with the persona's work or personal goal. Each value can assume numbers in the interval [0,3], where, considering Duration and Frequency, 0 means none, 1 means seconds, 2 means minutes and 3 means hours or longer; by considering instead Demands and Goal conflict, values mean none, low, medium, high, respectively. In order to compute the final usability for a task, average values of all the parameters are taken across all personas carrying out the task in question and summed up.

With the introduction of countermeasures mitigating the task's related risk, the usability will be affected. The properties and their relative usability components for the countermeasures are exactly the same as the ones just described, with the only difference that now they can assume values in the range [-3, 3]. Negative values are assumed when the countermeasure is helping to increase usability, positive values instead tell us that the countermeasure is decreasing the usability. Again, average values among countermeasures' usability values are taken across all the countermeasures affecting the task in question, they are summed up and added to the previous computed usability value. As result, the higher the score the less usable is the task for the associated personas and vice versa.

A security analyst, during the system design phase, is supported by a practical CAIRIS GUI when evaluating, quantifying and assigning risk and usability scores to resources and tasks, translating the quality attributes, that the analyst specify, in numbers identifying the corresponding risk and trust score, thus addressing [Req.2.5](#). Clearly, an appropriate storage system has to be adopted, in order to contain those informations; in particular, CAIRIS relies on a MySQL relational database for the management and the access to informations such as assets, threats, vulnerabilities, risks, and usability informations such as personas and task descriptions, addressing, in this way, [Req.2.1](#), [Req.2.2](#).

In order to successfully retrieve the data stored within the CAIRIS database, apposite APIs, have been implemented within the application, allowing the CAIRIS server to be contacted. The exposure of these APIs allows external clients, that have been authenticated beforehand, to retrieve such informations, addressing [Req.2.6](#) and [Req.2.7](#).

Another important feature of the IRIS framework, over which CAIRIS has been built, is the way according to which it provides a visualisation schema for the Risk and Task Analysis. Since, when creating a model, stakeholders and developers need to intuitively read it and interpret it, colours are associated to tasks, threats and vulnerabilities according to their usability or risk value. Red is devoted to risk while blue is devoted to usability; the more intense is the hue, the higher is the risk or usability value. The authors in [28] explain that adopting this approach is helpful in reducing visual clutter, especially when models grow large. In this way, the model provides a quick and easy view of the analysis.

### 4.1.3 Novelties

CAIRIS is the only Requirements Management tool able to support a software design process by dealing with both usability and security, exploiting the fact that both security and usability models share the concept of Requirements. There exist, however, a few other tools which instead integrated security aspects in Requirements Engineering models, in order to meet with the necessity of developing secure software design techniques.

There are tools which focus on specifying requirements, others that allows the definition of security aspects related to requirements, but still with naive and simplistic approaches, others focusing on modelling and others that focus on performing security analysis. Although CAIRIS origins are in specifying requirements, it has been extended in the years to be a tool which does all of the jobs described plus giving a practical way of managing usability requirements too, which it seems to be quite an innovation due to the current lack of tool-support allowing analysts and developers to inform security design with this kind of usability insights.

The desire of having a tool which accounts both for usability and security comes from the fact that the time and the costs, needed to implement a method that includes those features inside a traditional requirements analysis, are higher in respect to just specifying business requirements, since requirements engineers generally are not trained in eliciting security requirements and they need to rely on security experts in order to do so, as addressed by [62]. In this section, Requirements Management tools which have been enhanced to support security or usability

features will be reviewed, highlighting differences in respect to CAIRIS.

### STS-Tool

STS-Tool [73] is a security requirements modelling tool, which, similarly to CAIRIS, is able to present multiple perspectives of the system designed; in particular it allows three complementary views: social, information and authorization view. In this way, different interactions among actors can be analysed by working on a specific perspective instead of building a single big model of the system at hand, helping designers separating concerns.

The social view focuses on a goal-oriented modelling. Concepts of *Agents* and *Roles* are introduced; the relationship, which is established among these two entities, is based on the fact that an agent plays a role in order to fulfil a certain goal. Both *Agents* and *Roles* are identified by the term *Actor*. The goal model ties together goals and documents, which represent informations that *Actors* own, or manage in some way. Another important concept, in the social view, is the *Security need*: security properties are assigned to social relationships between *Actors*, in particular they can be assigned to goal delegation relationships (an *Actor* transfers a responsibility to another) and to document transmission relationships (an *Actor* allows the transmission of a document to another in order to accomplish a goal). The security properties that can be specified are the fundamental security properties we account in literature: integrity, availability, confidentiality, authentication, trustworthiness, non-repudiation.

The information view, instead, is used to go deep inside the informations that documents contain and the relationships that exist among them, indeed pieces of information belonging to different actors can constitute, if put together, another piece of information, which can be of some value, therefore, for this reason, they need to be properly protected.

The third view is the authorization view, defined to determine if informations are exchanged and used in compliance with confidentiality restrictions. In other words, when an *Actor* shares a document with another *Actor*, it has to define which authorization properties is transferring with the document, which can be: read, modify, produce and transmit.

Finally, when the models are ready, the tool gives the possibility to perform three kinds of analysis: a well-formedness analysis, a security analysis and a threat analysis. The purpose of well-formedness analysis is to verify whether the diagram built by the designer is consistent and valid according to the modelling language over which the tool is built (STS-ml [65]); security analysis, instead, is performed to verify whether there is any possibility for the *Security need* to be violated; it takes into account the semantics of STS-ml, looking at the behaviour of the rules' propagation in transmission relationships. What could happen, for example, is that, in the design phase of the model, we allow an actor to transfer certain security properties, associated to a certain document, that the actor is not allowed to transfer, actually, because he does not possess them neither. Finally, the threat analysis focuses on events threatening goals or documents, as well as social relationships involving goals and documents. Starting from the specified threats, it analyses how they propagate their impact over goal trees, documents and social relationships, in order to be able to see, at the end, all the resources that could be vulnerable.

STS-Tool allows, however, to define and associate threats to goals, documents and social relationships but in a very simplistic manner, since it assumes that the represented events threatening assets and the assets they threaten are the result of a risk analysis, performed with other specific tools.

No concept of persona is introduced in STS-Tool; *Agents* refer to concrete known participants of the system but they have no general characteristics or behavioural attitudes associated, that define a persona's profile. So, actually, they are just simple definitions of goals' and documents' owners. For the same reason, attackers profiles don't exist.

Without the persona's definitions, it becomes harder to identify possible threats arising in different

scenarios, indeed there is no concept of environment supported by this tool. Furthermore, there is no concept of Misuse Cases [70], which are used to document negative scenarios, where goals aim to threaten the system.

Also from the visualisation point of view, CAIRIS seems to be more effective, by using colours' distinctions in order to immediately give the perception of the risk's entity. Moreover, models, in CAIRIS, are automatically created after the requirement eliciting phase, while in STS-Tool have to be manually created, resulting in a much slower and error prone approach which undermine the intrinsic usability of the tool.

## **CORAS**

An attempt to provide a risk analysis framework has been done by CORAS [21]. CORAS is a model-based methodology, based on UML [58], which provides a customised graphical modelling language, to improve the efficiency of the analysis process and the quality of results. CORAS framework is tool supported and it provides a deep focus on threats and risks. Actually, the IRIS meta-model, over which CAIRIS has been built, is weaker in modelling risk, since it aims to a much broader vision in supporting secure software design, focusing on various elements in different contexts of use [31].

From the operational point of view, CAIRIS allows to: at first, elicit security, usability and functional requirements, then, visualise the model created, in order to have a quick and direct view of possible issues to be managed during system design process. Differently, CORAS provides a way of manually generating a risk model, starting immediately from the design phase, approach which is more prone to forget important aspects that should be taken into consideration, and putting, in this way, a lot of confidence in the security analysts who are in charge of creating the model.

Moreover, in CORAS, there is no concept about context of use; if a slightly different context has to be analysed, a new model has to be created. CAIRIS, instead, by introducing the notion of environment, supports a more various and deep analysis of different scenarios that could be of interest, automatically adapting the model. The idea underpinning CAIRIS is that risk can change according to the scenario, as well as the security properties of assets.

## **SeaMonster**

A tool, similar to CORAS, which has been developed to support risk analysis is SeaMonster [64] [51], which is a graphical modelling application allowing the description of different aspects related to security. More specifically, it allows to create models from three different perspectives: what causes the vulnerabilities, how the system is threatened and can be attacked because of the vulnerabilities, how the vulnerabilities or their effects can be mitigated through countermeasures. Again, this is a simple tool which only focuses on building risk oriented models which can be very sophisticated, but not really scalable if models grow up, since they have to be created drawing all the modules and the relationships by hand. CAIRIS, on the other hand, allows the visualisation of even large and complex models just by automatically elaborating the relationships between the different components.

## **Rational DOORS**

There exist other kinds of tools, instead, which were especially built supporting requirements engineering. Rational DOORS [34], for example, is a Requirements Management tool developed by IBM. What Rational DOORS developers claim is that poor and improper requirements management is the single biggest reason for project failures and can lead to disconnected working

teams, waste of time, out of control costs and unhappy customers.

Rational DOORS is a tool for capturing, tracing and analysing requirements to be managed during the developing of complex systems; however, aside from the functional requirements aspects, which are certainly dominant in the tool, an interesting feature of Rational DOORS is that it is open to further extensions, thanks to the DXL (DOORS eXtension Language) scripting language, customised to Rational DOORS, which, together with the Rational DOORS C APIs, allows a simple integration of the tool with other applications. This means that the integration with other existing tools which have functionalities that can be compatible with Rational DOORS is possible. Although Rational DOORS seems to be a well accepted tool in the area of requirements engineering, it doesn't support any graphical representation of requirements, for this reason authors in [74] present a way of extending DOORS introducing a DXL plug-in, enabling graphical modelling within DOORS. Probably, those good flexibility properties, that allow the simple integration with other tools, could be exploited to enhance Rational DOORS to include security and usability requirements plugins and risk analysis mechanisms, aiming to propose Rational DOORS as a good security and usability requirement tool.

## 4.2 SAFAX

### 4.2.1 Introduction

SAFAX [40] is a XACML-based architectural framework tailored to the development of extensible authorization services for distributed and collaborative systems, like cloud storage systems. Since cloud storage services are recently becoming a dominant component in people's everyday life and are often already integrated in mobile or desktop devices they possess, it is quite usual to be signed to different cloud providers and to have contents spread all over those different storage systems. Several issues arise in such a situation: most of the time, cloud providers adopt identity-based access control mechanisms, which means that for each specific user or group of users, a specific policy has to be defined. The problem, from the point of view of the person which is in charge of defining the policies, is that, when specifying some rules for a group of users, all of the single individuals of the group have to be known and inserted in the list, which is something infeasible when groups are composed by a large number of users. At the same time, from the point of view of the system itself, all the users have to be necessarily registered on it, otherwise there is no way to identify them and to guarantee that the correct policy specifications are applied. Another not negligible issue in the policy's definition process in those cloud storage systems is the limited expressiveness capability: generally, it's not possible to define complex and dynamic rules that capture parameters from the context (for example time, location, connection security properties, risk) and define policies which are not simply based on the specifications of users, roles and actions over objects. Moreover, policies specification's methods are not at all standardised, each cloud service provider implements its own way to define policies: some of them provide a GUI through which insert parameters, like the user's email and the actions he can perform on the object, some others instead require the knowledge of a specific language; this can have a strong impact on users' workload, due to the fact that they need to learn how to correctly deploy all of those policies. From this issue, consequently, another one arise: if we need to update the kind of access a certain user has granted over resources sparse among different cloud providers, there is no way of doing that once and for all, but we are forced to manually modify the policies on all the cloud domains. Furthermore, not only policies specification's methods are cloud provider's specific, but generally, also the semantic that each system use is quite peculiar. The same term used in a system can be present in another with a slightly different meaning, creating confusions and misalignments. Those are the issues that SAFAX aims to solve.

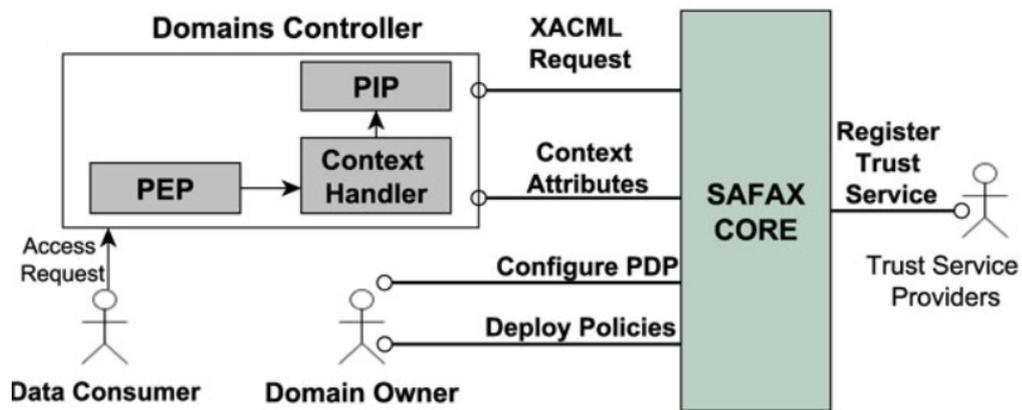


Figure 4.2. SAFAX framework [40].

#### 4.2.2 Baseline Architecture and Goals

From the point of view of the baseline architecture, SAFAX relies on the XACML standard. In particular, the main features of SAFAX stand in how the XACML components relate with each others and with, eventually, other possible external components.

The SAFAX framework consists of three main blocks: domain-specific components, SAFAX-CORE and trust services (see figure 4.2). PEP, CH and PIP are domain-specific components and are generally implemented by the domain controller, which is the cloud service provider itself, so they depend on the application environment, addressing [Req.1.1](#), [Req.1.3](#), [Req.1.5](#). The only requisite they should have is being able to handle the conversion between the attributes given by the application and the attributes expressed in the XACML format. The SAFAX-CORE is the main part of SAFAX's implementation and includes several components: first of all, the Router, which receives the XACML requests and context attributes through specific interfaces, and forwards those requests to the PDP. In the SAFAX-CORE, a dedicated PDP is assigned to each Domain Owner, thus addressing [Req.1.2](#), and it is identified by a PDP-URL, in such a way the Router is able to identify the correct PDP to contact for an access request. When a PDP receives an access request, it fetches the policies associated to the Domain Owner from the PAP service.

The PAP module built in the SAFAX-CORE is unique, indeed each Domain Owner of each Domain Controller relies on the same PAP. This is one of the key point of SAFAX: to offer a single point for deploying and managing policies, addressing [Req.1.4](#). PDPs need also to be configured through a PDP Configuration module (PDPC), thanks to which it is possible to define, for a Domain Owner, the kind of PDP implementation to use and configure the selected PDP with the proper parameters (like the root combining algorithm). A schema for the existing relations between components within SAFAX-CORE is provided by figure 4.3.

A Trust Service Repository is also present in the SAFAX-CORE and allows external service providers to register their services within SAFAX; moreover, it allows to any service already registered to consume other registered services. In order to successfully complete the registration, external services need to have assigned a Service identifier and to provide the Service Provider's identity, a Service description, the service endpoint (URI), the HTTP method to use to invoke the service, the Request parameters data type, the Response parameters data type, the Request and Response messaging format and the Service Type, which indicates the kind of service is provided (PDP, PAP, PEP, CH, PIP, External Trust Service). It's crucial that external services' APIs must conform to the specifications of SAFAX in order to communicate with the system. In order for these trust services to be exploited, SAFAX's developers chose to enhance the PDPs with an External Service Extension module, which is a component that handles the invocations to external trust services, in order not to disrupt the existing components.

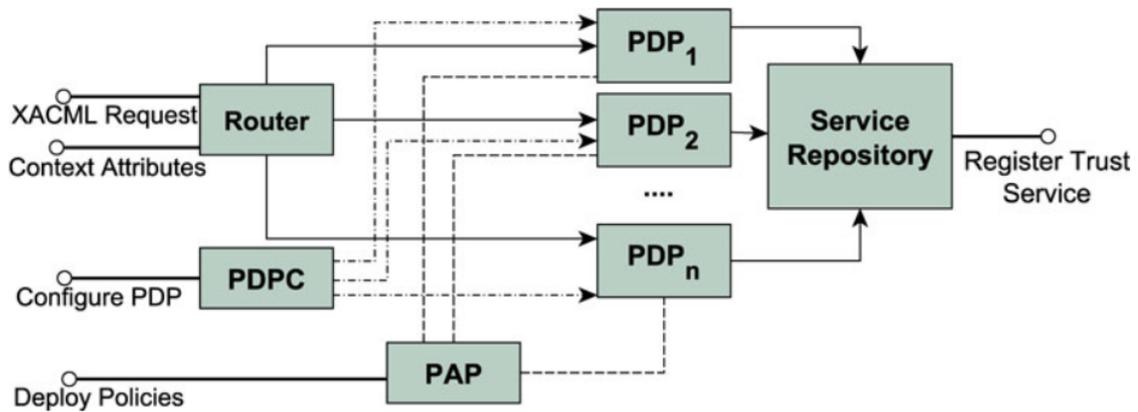


Figure 4.3. SAFAX-CORE inner relations [40].

By exploiting those trusted registered services, it is possible to define User Defined Functions (UDFs), which are Domain Owner's custom functions and are used to retrieve external parameters to include in the access control decision.

The SAFAX architecture does not make any assumption on the trust model to use in the cloud environment; it has been built to work in the same way either with a Centralized Trust model or a Distributed Trust model or a Distributed and Delegated Trust model, see figure 4.4.

According to a Centralized Trust Model, the Cloud Provider controls all the authorization services. The domain-specific components, the SAFAX-CORE and the trust services are within the Domain Controller, which means the Domain Owner puts his complete trust in the Cloud Provider for the storage, the correctness of the access control decisions and the enforcement of the policies. This kind of approach can be suitable to those environments where it is too risky to outsource some services, like in military environments.

In a Distributed Trust Model instead the Domain Owner has to put his trust in two different entities: the Authorization Service Provider, which is composed by the SAFAX-CORE and the trust services, and the Domain Controller, which, from its side, implements PEP, PIP and CH. This means that, still, the Domain Owner cannot exploit services which are external to the authorization provider but, now, he can connect several Cloud Storage Services to the same Authorization Service Provider, exploiting the property of SAFAX to be a unique point where to deploy policies.

The third model, the Distributed and Delegated Trust model, is the most general one; the Domain Owner has to trust three different entities: the Domain Controller, for the secure storage, the Authorization Service provider, for taking correct access decisions and deployment of policies, the External Provider, for returning correct and accurate informations to be used in policy evaluation. Adopting SAFAX, as a unique point where to deploy policies, relieves the user, that needs to update some permissions, from the burden of changing those permissions specifically on every Cloud Provider. Moreover, by adopting XACML, which is an attribute-based policy language, SAFAX gives the possibility to extend policies with custom attributes, improving the limitations of Cloud Providers in defining policies and it defines a standard way to define them, solving in this way the problem, for the user, to learn and adopt different languages. Moreover, by using UDFs, SAFAX provides not only a way of defining fine-grained policies, using informations and attributes that are taken from External Trust Services, but it also relieves the system from having to know specifically every user identity, addressing this problem to the outside external services. It is also possible to exploit an External Trust Service which is able to align semantics of terms, among Cloud Providers, avoiding in this way vocabulary interpretation's problems that otherwise should be solved with human intervention.

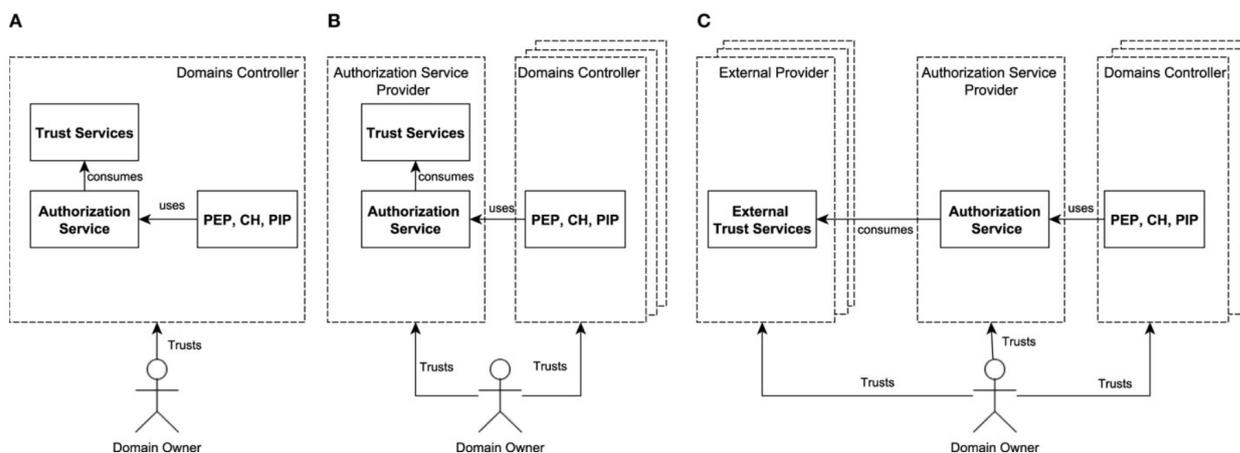


Figure 4.4. Trust models supported by SAFAX (A) Centralized trust model. (B) Distributed trust model. (C) Distributed and delegated trust model. [40].

### 4.2.3 Novelties

There are works that aim to improve XACML components from the point of view of performances, like [46], where XEngine, a scheme for efficient XACML policy evaluation based on the numericalization and normalization of XACML policies, is proposed; or in [49], where clustering techniques and reordering algorithms are used to improve access control to policies.

SAFAX, does not bring any novelty neither for what concerns policy evaluation, since it just reuse already existing XACML implementations, nor for what it concerns performances. The main novelty, that SAFAX introduces and makes this tool unique, lies in the service oriented architecture adopted for policy evaluation. The first thing to observe is that every component of the XACML reference architecture is implemented as a loosely coupled service. Loose coupling has a positive connotation, since it implies that services share a limited amount of features and therefore they can evolve independently [56]. This is a great advantage, since when we need to update and modify some properties of a service, we can do it, without having to adapt all the others accordingly.

By looking at the literature, some work has been done in designing loosely coupled modules inside authorizations systems, like in [44], where an extensible XACML authorization web service has been proposed, which only focus on the development of an extensible PDP, enhanced with some additional modules able to extend policies with non-standard data type; or in [47], where, again, an authorization system architecture is described, highlighting the fact that PDP implementation is refrained from closely coupling with the context-handler component.

The main difference, in respect to those approaches, is that SAFAX does not focus only on the PDP, but on all the components of the three main blocks: domain-specific components, SAFAX-CORE's components, external trust services are implemented as loosely coupled services. In particular, the External Service Extensions component, which is the one that extends the SAFAX's PDPs, implements, for each UDF, a self-configuring client, that consumes the external trust service; indeed, when the Domain Owner specifies policies that require additional trust informations to make a decision, the PDP service contacts the external trust services. By doing this way, in a Distributed and Delegated Trust model, is the Domain Owner who is in charge of determining from which source the informations used in policy evaluation have to be fetched, allowing in this way the maximum of flexibility and governance to the user who wants to store and protect his contents.

We can say that SAFAX breaks the limits imposed by current XACML implementations, which are monolithic and does not provide proper functionalities for cloud environments, where, instead,

a dynamic approach is needed, different access control parameters have to be checked and policies have to be enhanced with more details.

The fact that external services can be easily added to SAFAX, by registering to the SAFAX Trust Service Repository, is another relevant feature of the tool. Some work has been carried out in order to address the problem of the limited support for establishing and maintaining a web of trust, in particular focusing on cloud federations. In [43] we find a definition for cloud federations: “Cloud federation comprises services from different providers aggregated in a single pool supporting three basic interoperability features - resource migration, resource redundancy and combination of complementary resources”. In [10] an inter-cloud federation architecture, from an high level point of view, is presented: a Cloud Exchange orchestrator acts as a market maker, aggregating the infrastructure demand of users and the available infrastructure supply of the Cloud Providers, that adhere to the federation. In this way, the architecture couples the administratively and topologically distributed storage and computes capabilities of Clouds as parts of a single resource leasing abstraction. For sure, some advantages can be seen from this perspective: from the point of view of the user, the choice of the Cloud provider will be made according to the market rules, dynamically, in order to benefit from the best QoS conditions guaranteed; from the point of view of the cloud providers, instead, advantages come from a better resource’s allocation and migration of workloads, improving the usage and the management of storage and computational resources. If the model just described is then enhanced with the security and trust model for Cloud infrastructures proposed in [14], it is possible to obtain a trusted cloud federation where users’ credentials can be reused from one provider to the other, guaranteeing, in this way, the secure access to resources, stored in different clouds, without having to be registered on each cloud domain.

A slightly different approach is proposed in [5], where access control modules are now distributed among different providers and not centralized in a single entity; the federation is then kept together by Service Level Agreement (SLA) modules which forward and evaluate the permissions transferred between different providers.

By adopting the previously described approaches, the problem, which involves both authentication and authorization, is that, when we need to include in the Federation a new cloud provider, we need to update all the components of all the other already existing services, in order to maintain this web of trust. The workload, resulting from this process, becomes more demanding the more cloud domains are present. SAFAX, instead, offers a simple way of increasing the number of trusted services just by registering on the SAFAX Trust Service Repository. In such a way, on the one hand, SAFAX relieves the user from the burden of having to specify and manage different access control policies on each cloud platform, on the other hand, it helps cloud providers to reduce the implementation and management costs of developing their own authorization solutions by outsourcing it.

In order to summarize, the main novelties, that SAFAX introduces, stands in how XACML components and external services are connected together and in the way all components are implemented, which is as loosely coupled services, thus providing the flexibility, extensibility, and scalability needed to manage authorizations in cloud environments. Moreover, by proposing itself as a unique point where to deploy policies belonging to several registered services, SAFAX improves the usability perception of people that have to manage those services and helps the providers of the same services which, registering to SAFAX, wouldn’t need to implement their own access control systems any more.

# Chapter 5

## Integration

In this chapter, we present the high level implementation of the final framework obtained with the integration of SAFAX and CAIRIS tool, presenting the complete process to carry on when configuring such a system, in order to be able to exploit a risk-based decision mechanism when evaluating access requests. Then, details about the architecture and the components that characterize it are given. Finally, some considerations about tool's performances are outlined.

### 5.1 High Level Implementation

As a result of this thesis work, the implementation of a risk-based authorization mechanism has been extended to a context-aware scenario, where decisions are tailored to the specific environment the authorization system is facing.

This means that the authorization mechanism should be able to distinguish among requests that involve same parameters but in slightly different contexts. A context of use can represent a time range (Day, Night, from 10 am to 12 am, and so on) or a geographical concept (access request coming from an internal machine of the organization or from the outside), or, in general, any specific scenario the authorization system should be able to deal with. Adopting a context aware mechanism let the dynamic of the decisions to improve; indeed, by providing to the authorization mechanism a set of parameters to be evaluated, in order to take an authorization decision, we may retrieve different risk scores, depending on the specific scenario in which those parameters are considered. By supporting a mechanism of this kind, it is possible to address a common issue, observed in several works that have been revised in the state of the art analysis performed in Chapter 2, which is the failure of the attempts, that have been made, in identifying risks within specific scenarios of usage, obtaining, in this way, quite static approaches which don't address the real need for a precise and dynamic way of creating a context able to describe the reality, in all its characterizing facets.

We can identify the definition of contexts as a first necessary phase in order to consider usage situations from different perspectives and elicit those scenarios who are of interest and those who aren't. By distinguishing among them, it is possible to improve the flexibility and the accuracy of the system which relies on those scenarios, separating concerns, subjects, resources and vulnerabilities involved. For this reason, proceeding with a first general outline of the environments is the first step to perform towards the categorization of risks.

The second necessary step would be then, for each identified environment, to build an appropriate risk model, in order to configure the system. To this purpose, CAIRIS, exploiting the fact of being a security and usability requirements management tool, proposes itself as a smart and functional way to accomplish this task, allowing a user to create an a priori detailed risk model, tailored to

the business scenario the risk-based authorization system will be used for.

In order to achieve this, CAIRIS has to be exploited in its full potential. Assets, risks, threats, vulnerabilities, personas and possible attackers can be defined, in such a way to feed the tool with all the necessary requirements to automatically create the model. In order to exploit quantified risk notions, for each specific identified environment, threats have to be assigned to the assets they threaten and vulnerabilities have to be assigned to the assets they expose. Moreover, for each threat, the likelihood with which it will manifest itself and the asset's security properties it will endanger have to be specified, together with the vulnerabilities' severity.

By putting together those informations, the risk, for a certain resource, is going to be automatically computed by the system making the product between likelihood of the threat, severity of the vulnerability exposed by the risk and the risk impact. The latter represented by the security properties endangered for the asset.

In this way a risk score is assigned for each asset in the model, according to the insights presented in [28].

After these preliminary but fundamental “design” phases, a database of risk's notions will be present and ready to be used.

At this point, a user, that wants to evaluate some risk policies, just have to complete the registration on SAFAX service, in order to obtain an account. A preliminary configuration phase is needed: first of all, the user should create a project, then he can proceed with the creation of the demos associated to the project. According to [40], a demo can be defined as “the management point for regulating access to the data”; a demo allows the user to deploy XACML policies, upload requests, choose the root combining algorithm to use inside policies and configure the kind of PDP implementation to use in the evaluation process, indeed a single PDP is associated to each demo. Furthermore, together with the aforementioned features, thanks to this thesis work, the user is going to have the possibility to set his own CAIRIS credentials and database informations to use in order to successfully authenticate on CAIRIS and retrieve risk informations (see the User Manual in [Appendix B](#) for a more detailed description of the steps necessary to configure and use SAFAX tool).

Once the setting of these informations is done, the user can exploit the authorization service provided by SAFAX, both via SAFAX GUI, in order for policies and requests to be easily tested, and also by configuring a third-party software system, that access SAFAX programmatically. The latter method is the most interesting one because it allows a third-party system to contact SAFAX via proper APIs and, after having performed an authentication procedure, gives the possibility to contact the PEP web service, provided by SAFAX, and consume the authorization service, by submitting a valid request. The same approach can be used to remotely deploy policies. Obviously, before those operations can successfully work, the third-party system should be first configured on SAFAX, in order to obtain authentication credentials and a dedicated PDP, according to the specification given in [59].

## 5.2 Architecture

As we saw, SAFAX is an extensible XACML-based architectural framework for policy evaluation. It can be used to evaluate simple policies that adhere to the XACML standard, moreover, at the same time, it can be easily extended to some customized way of treating policies, by means of UDFs. UDFs are User Defined Functions and, if they are invoked in the *Rule* part of the XACML policy, allow to integrate in the evaluation process a new customized way of treating attributes. By implementing those UDFs with the ability of treating a risk concept, it is possible to extend this framework to support a policy evaluation process based on risk.

The service, that has been considered of particular interest in order to provide the risk values to use during the evaluation process, performed by SAFAX, has been identified in CAIRIS. As

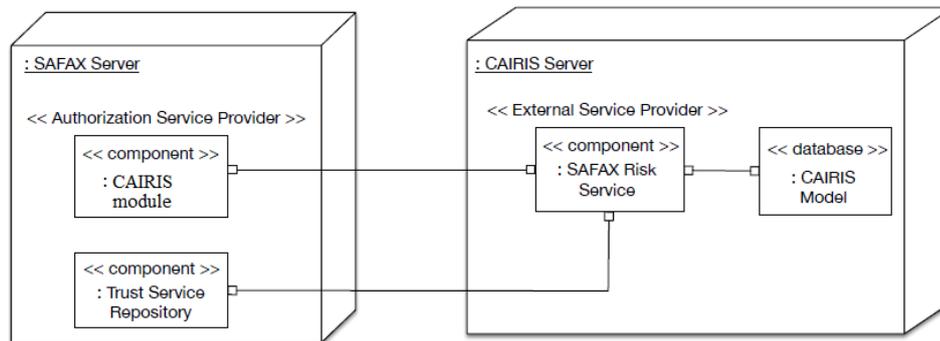


Figure 5.1. High level architecture supporting the CAIRIS-SAFAX interaction.

we saw, CAIRIS is a security and usability requirements management tool, which allows the characterization and construction of real case models supporting the system design process.

After having presented how these two tools successfully conform to the first two classes of requirements, identified in Chapter 3, necessary to develop a risk-based access control framework and involving the implementation of an XACML-based architectural framework and of a risk estimation module, we are going to present how the integration of the two has been successfully obtained.

The architecture that allows the interaction among the two services, CAIRIS and SAFAX, is shown in figure 5.1. The *SAFAX Risk Service* component is a RESTful web service and runs on a CAIRIS server. Logically, the component contains a controller object, which has been placed behind the APIs, and a DAO object, which works as an intermediary between the controller and a database proxy. The *SAFAX Risk Service* interacts with a *CAIRIS Model*, which is a relational database, containing security informations about the assets, threats, vulnerabilities, risks, and usability informations such as personas and task descriptions, addressing the [previously](#) identified need for managing those informations.

This service is registered within the *Trust Service Repository* on the SAFAX server and is accessed, by means of specific APIs, by the *CAIRIS module*, which has been the main development focus of this thesis work, and runs on a SAFAX server, addressing [Req.3.4](#).

The *Trust Service Repository* uses a MySQL back-end database to store the informations regarding a service, such as the URL to contact the service, the HTTP method to use, the number of parameters to include, the kind of service provided (PIP, PEP, UDF, etc). Moreover, in order to support the [aforementioned](#) configuration procedure, to exploit the CAIRIS service, two brand new tables have been added to the database: a *sfx\_cairis\_credentials* table and a *sfx\_cairis\_db* table (see the Developer Manual in [Appendix A](#) for more details about these tables). The first one is used to store the CAIRIS credentials, specific for each demo present in the system, so a reference to the *demoid* had to be made in order to successfully perform the association; the second one, instead, stores the informations about CAIRIS databases that can be contacted to retrieve risks. The logical relation linking the two tables stands in the possibility of having multiple databases associated to the same credentials. From CAIRIS side, indeed, each user has the possibility to create his own databases, related to the corresponding developed models, which are personal and dedicated to the user.

The *CAIRIS module* is the entity responsible of managing the communication between SAFAX and CAIRIS tools, in particular between the SAFAX PDP and the *SAFAX Risk Service* implemented by CAIRIS. To this purpose, the *CAIRIS module* intercepts the PDP invocation of the CAIRIS risk service, during policy evaluation. The risk service, indeed, is summoned by means of a specific set of APIs, that the PDP itself invoke, in order to contact the module and that represent a risk

request, addressing [Req.3.3](#). Then, the *CAIRIS module* constructs a valid HTTP request, by filling it with all the necessary parameters to retrieve the risk score, in such a way to cope with the format requirements supported by the CAIRIS APIs and thus addressing [Req.3.1](#). In order to retrieve these risk values, the *CAIRIS module* has to go through two preliminary steps: first authenticate on CAIRIS system, then contact the desired CAIRIS database. Basic HTTP authentication is the implemented mechanism used to obtain a valid session on CAIRIS; this session is a unique identifier for an incoming request and it is represented by a string of 32 alphanumeric characters (see [table 5.1](#) for the details about the API used). The session obtained will be then used when contacting the *CAIRIS Open-Database API*, used to open the specified database (see [table 5.2](#) for more details about the API).

According to the developed solution, a user, that wants to evaluate a request against a policy, exploiting the notion of risk, has the possibility to specify, in it, one out of four possible combinations of attributes, in such a way to retrieve the desired risk value. These four combinations are mapped into four different APIs supported by CAIRIS (details about these APIs can be found in [5.3, 5.4, 5.5, 5.6](#) tables):

1. `/api/risk_level/asset/{asset_name}`, see [table 5.3](#).
2. `/api/risk_level/asset/threat_type/{asset_name}/{threat_type_name}`, see [table 5.4](#).
3. `/api/risk_level/asset/{asset_name}/environment/{environment_name}`, see [table 5.5](#).
4. `/api/risk_level/asset/threat_type/{asset_name}/{threat_type_name}/environment/{environment_name}`, see [table 5.6](#).

By using the first enumerated API, a user needs to specify, in the request, the asset name, which identifies a resource. In this case, CAIRIS will return as a result the highest risk value associated to the specified asset.

If, instead, a user wants to exploit the second enumerated API, he needs to specify the asset name and the threat type name, which represents a possible threat associated to the asset. In such a situation, CAIRIS will return the highest risk score associated to the resource given the specified threat type.

With the introduction of the last two enumerated APIs, we allow the further specification of the environment as a context of use, in such a way to let CAIRIS return, respectively, the highest risk score associated to a resource in the given environment and the highest risk score associated to a resource, given a certain threat type, in the given environment. In this way, we cope with the need, identified at the beginning of [Section 5.1](#), for distinguishing among risks involving threats and assets which belong to multiple environments.

In order to find a matching for resources, threat's specifications and environments, clearly, the two tools need to be aligned, using the same identifiers.

Beside the URL parameters, the only data parameter needed in these APIs is the identifier of the session, which will automatically point the risk request towards the previously opened CAIRIS database.

Once the desired risk value has been retrieved from CAIRIS, the *CAIRIS module*, present in SAFAX, parses the response and evaluate it, then it communicates to the PDP the final decision, so that the PDP can enforce it.

The evaluation mechanism implemented is threshold based, meaning that, in order to take an authorization decision, the risk value is compared against a threshold, but, could be, eventually, easily extensible with other mechanisms. When the risk value is higher than the threshold, it means the risk is too high to be accepted and the authorization request cannot be granted, otherwise, when the risk value is lower or equal than the threshold, it means the risk is acceptable and the system should grant the authorization to the request. With the definition of the evaluation logic

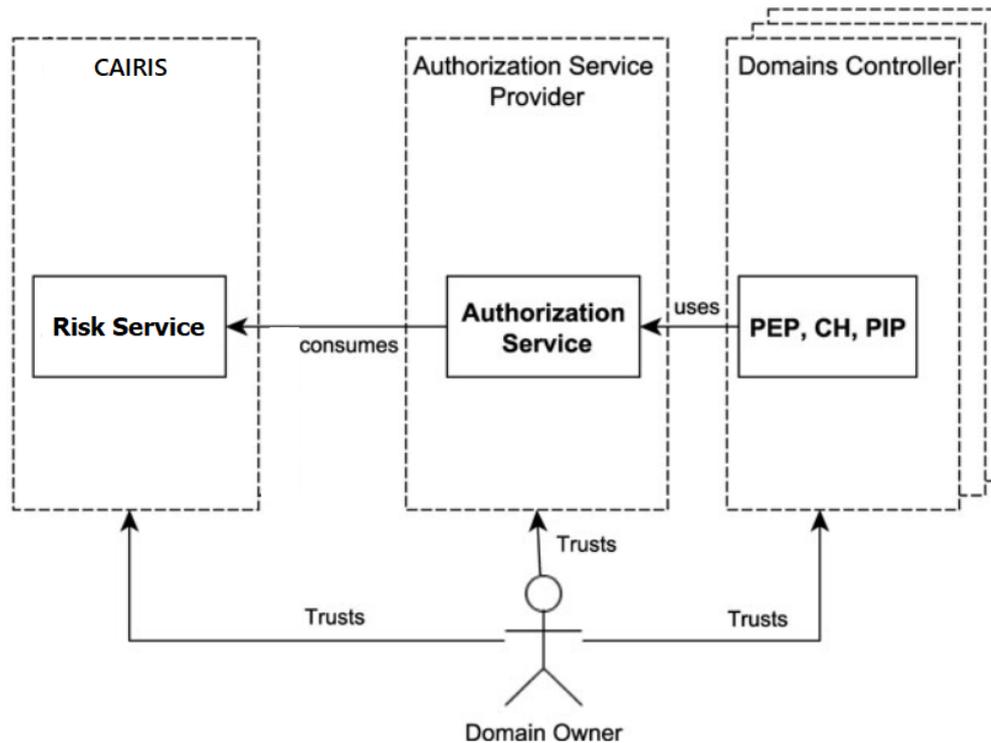


Figure 5.2. Distributed and Delegated Trust Model with CAIRIS usage.

we can address [Req.3.2](#).

Thresholds have to be specified in the *Rule*'s condition, within the policy, in such a way that we can easily associate them to the specific resources they refer to. Therefore, when defining a policy, a user, together with the asset, threat type, environment attributes, has to immediately define a value for the resource's risk threshold.

As a result, the current implementation allows to establish a distributed and delegated trust model, the most open and extensible among the trust models identified by [40]. This model provides the highest level of flexibility, by delegating to three different parties the tasks. The SAFAX service acts as the Authorization Service Provider, while CAIRIS is the External Provider, which provides an external trust service. The Domain Controller role will be identified by any application which decides to consume the SAFAX service. Figure 5.2 shows the distributed and delegated trust model which involves the interaction with CAIRIS as an external service.

### 5.3 Performance Considerations

As revised in Section 4.2, one of the most relevant features of SAFAX is the way it has been built. The architecture, indeed, includes several components which have been implemented as loosely coupled services, thus allowing to anyone, who would like to exploit the service provided by a customized XACML component or external service, to easily include it inside the project. It follows that great extensibility properties come from the intrinsic core architecture of SAFAX, emphasising, in this way, one of the main goals of the developers of the tool, which is to create an application that could hugely evolve in the future, together with technology innovations and services, supporting advanced authorization scenarios, like papers [25], [48], [75] present. SAFAX services have been implemented as RESTful web services that strictly conform to the

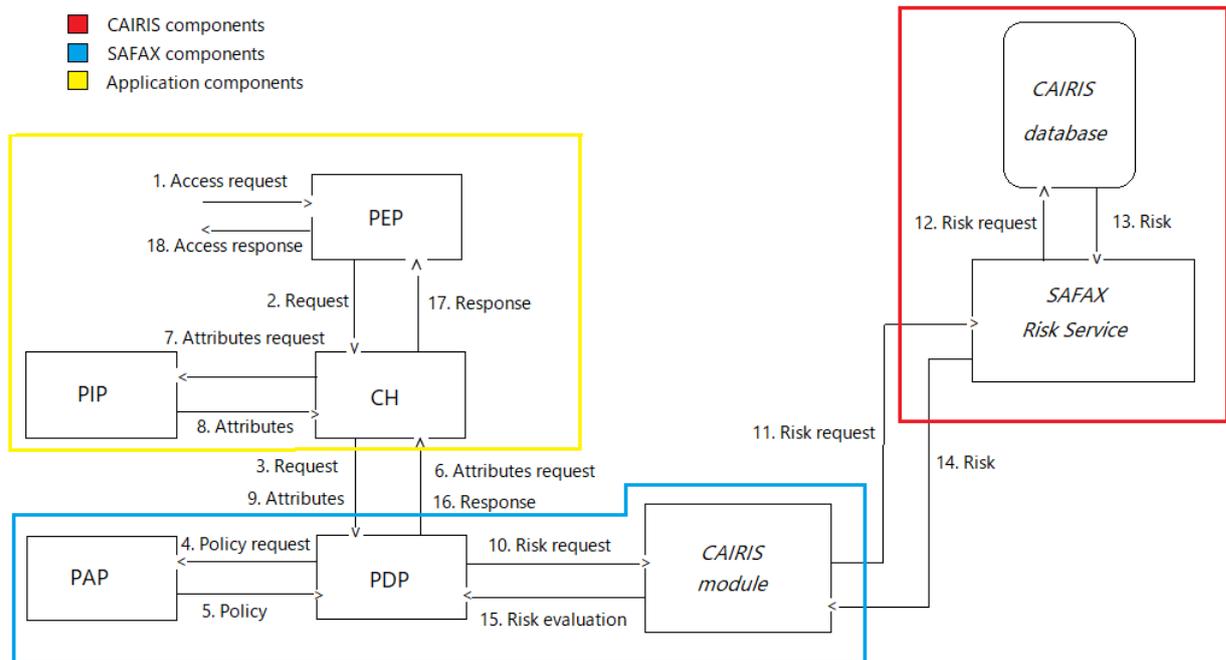


Figure 5.3. Information flow during a risk evaluation request, once preliminary authentication steps have been performed.

properties of being purely stateless with all information related to the state contained either with the request or stored in a persistent database. Moreover, services communicate with each other either in JSON or XML [40].

It is important to make those premises in order to better understand the functioning of the tool and the results a user should expect from it, since the underlying architecture, over this tool has been built, strongly determine its performances.

Due to the inherent structure of the framework, in order for the modules to communicate with each other, several HTTP calls have to be performed, sending requests and retrieving responses in every communication step among the XACML reference architecture's components, which act as independent standalone web services. It follows that, whenever a request is submitted and have to be evaluated against a policy, first of all, the PEP web service is contacted through the proper API, the PEP forwards the request to the web service acting as Context Handler, which forwards the request to the PDP and, eventually, if the PDP asks to, it fills the request with context informations taken from the PIP. The PDP needs to contact the PAP module, then, to retrieve policies. Moreover, during the evaluation procedure, the PDP contacts the *CAIRIS module*, by means of the proper APIs.

The *CAIRIS module*, then, needs to perform an HTTP request towards the CAIRIS server, in order to retrieve the risk information. Actually, before being able to retrieve the risk score from CAIRIS, an authentication procedure has to be successfully carried out and the desired database has to be opened in order to let the risk APIs point directly to the correct set of data. These are two distinct operations that happens in sequence. It would be certainly interesting to analyse the percentage of time those operations require, in respect to the total evaluation time for a request, in such a way to analyse the weight that these have within all the evaluation process and in order to have a clear understanding of the causes for all the experienced delays.

If an application relies on the XACML architectural components provided by SAFAX, exploiting the service programmatically, then the interface between the *CAIRIS module* and the *SAFAX Risk Service* represents the only point of communication between an internal SAFAX component

and an entity implemented outside the SAFAX server. If instead we suppose an application implements its own PEP, CH and PIP components, SAFAX becomes the Authorization Provider, offering the authorization features that PDP, PAP and *CAIRIS module* implement, according to the aforementioned distributed and delegated trust model. In this latter case there would be two communication flows crossing the public internet: the one representing the interface between the application specific CH and the SAFAX PDP and, again, the one between *CAIRIS module* and the *SAFAX Risk Service* on the CAIRIS server. The complete information flow happening in such architecture is shown in figure 5.3.

As a result, by avoiding to implement the whole authorization mechanism, offered by the SAFAX tool, as a monolithic component, a great extensibility is allowed, but performances are penalized: even if all the modules are physically deployed on the same server, logically they represent standalone web services that communicate as if they belong to different projects deployed on diverse servers, situation which could, in principle, be true, if again we take as example the distributed and delegated trust model where PEP, PIP, CH and external trust services are provided by different entities from SAFAX (figure 5.2).

A further interesting direction to take, when analysing the performances of the tool, could be to observe whether and how the evaluation time varies when the risk policy changes, in particular when it changes in respect to the different supported scenarios that exploit the four aforementioned APIs (tables 5.3, 5.4, 5.5, 5.6). The expressiveness of a risk policy changes according to these four situations: the most general case is represented by API 5.3, according to which the highest risk value for an asset, considering all the possible associated threats and within all the possible environments, is retrieved. In this case, the CAIRIS database has to compute and return the maximum value among all the risk scores associated to the specified asset, a task which is heavier the highest is the number of risk scores present in the database and associated to the specified asset, thus the highest is the number of threat for the asset and the number of environments in which it can be considered. By applying some sort of filtering, instead, either on the environment (5.5) or on the threat type (5.4) or on both (5.6), we would expect the risk retrieval process on the database to be a faster task, since the number of possible entries, over which the maximum operation have to be performed, should decrease in respect to the more general case.

<i>Session API</i>	Returns a valid CAIRIS session-id
URL	/api/session
Method	POST
HTTP Authorization Header	Type: Basic Base64Encoded: {username}:{password}
Success Response	Code: 200, Content: { <i>session_id</i> : String}
Error Response	Bad Request (400), Content: { <i>error</i> : String}

Table 5.1. CAIRIS Session API table.

<i>Open-Database API</i>	Allows a session-id to point to a specific CAIRIS database
URL	/api/settings/database/{ <i>dbname</i> }/open
Method	POST
URL Parameters	{ <i>dbname</i> }: String [Required] { <i>dbname</i> }: identifies the database to contact.
DATA Parameters	{ <i>session_id</i> : String}
Success Response	Code: 200, Content: { <i>resp_result</i> : String} <i>resp_result</i> corresponds to a message advertising the successful operation.
Error Response	Bad Request (400), Content: { <i>error</i> : String} Session Not Found (409), Content: { <i>error</i> : String}

Table 5.2. CAIRIS Open-Database API table.

<i>Risk API 1</i>	Returns the highest risk score associated to an asset
URL	/api/risk_level/asset/{ <i>asset_name</i> }
Method	GET
URL Parameters	{ <i>asset_name</i> }: String [Required] { <i>asset_name</i> } identifies a resource.
DATA Parameters	{ <i>session_id</i> : String}
Success Response	Code: 200, Content: { <i>risk_level</i> : Integer} <i>risk_level</i> corresponds to the risk score, where $0 \leq risk\_level \leq 10$ .
Error Response	Bad Request (400), Content: { <i>error</i> : String} Unauthorized (401), Content: { <i>error</i> : String}

Table 5.3. CAIRIS Risk API 1 table.

<i>Risk API 2</i>	Returns the highest risk score associated to an asset given a threat type
URL	/api/risk_level/asset/threat_type/{asset_name}/{threat_type_name}
Method	GET
URL Parameters	{asset_name}: String [Required] {threat_type_name}: String [Required] {asset_name} identifies a resource. {threat_type_name} identifies a threat type.
DATA Parameters	{session_id}: String
Success Response	Code: 200, Content: {risk_level: Integer} risk_level corresponds to the risk score, where $0 \leq risk\_level \leq 10$ .
Error Response	Bad Request (400), Content: {error: String} Unauthorized (401), Content: {error: String}

Table 5.4. CAIRIS Risk API 2 table.

<i>Risk API 3</i>	Returns the highest risk score associated to an asset in a specific environment
URL	/api/risk_level/asset/{asset_name}/environment/{environment_name}
Method	GET
URL Parameters	{asset_name}: String [Required] {environment_name}: String [Required] {asset_name} identifies a resource. {environment_name} identifies a context.
DATA Parameters	{session_id}: String
Success Response	Code: 200, Content: {risk_level: Integer} risk_level corresponds to the risk score, where $0 \leq risk\_level \leq 10$ .
Error Response	Bad Request (400), Content: {error: String} Unauthorized (401), Content: {error: String}

Table 5.5. CAIRIS Risk API 3 table.

<i>Risk API 4</i>	Returns the highest risk score associated to an asset given a threat type and in a specific environment
URL	/api/risk_level/asset/threat_type/{asset_name}/{threat_type_name}/environment/{environment_name}
Method	GET
URL Parameters	{asset_name}: String [Required] {threat_type_name}: String [Required] {environment_name}: String [Required] {asset_name} identifies a resource. {threat_type_name}: identifies a threat type. {environment_name} identifies a context.
DATA Parameters	{session_id}: String
Success Response	Code: 200, Content: {risk_level: Integer} risk_level corresponds to the risk score, where $0 \leq risk\_level \leq 10$ .
Error Response	Bad Request (400), Content: {error: String} Unauthorized (401), Content: {error: String}

Table 5.6. CAIRIS Risk API 4 table.

# Chapter 6

## Usage Scenarios

In this section, three usage scenarios will be presented in order to give some hints and suggestions about how the risk-based authorization system developed could be applied and used in real case situations. The first scenario involves a water company, which relies on a distributed system, both accessible via internal network or remotely from the outside, in order to manage the whole infrastructure over which the business has been built. In the second scenario, we present a possible application to an healthcare facility scenario. The third scenario involves a grid-based collaborative system, used by research centres to share knowledge and research progresses mainly about neural diseases, involving people's sensitive informations.

### 6.1 Scenario 1: ACME Water

ACME Water [2] is a UK water company using a cloud-based enterprise SCADA system. The company has been object of a case study [30] where a user-centered approach was taken to elicit and analyse information security policy requirements following reports of the Stuxnet worm [33]. To this purpose, ACME Water has a User Experience (UX) team which has carried out users research, by interviewing people, within the company, in order to create a database of personas characterizing typical roles in the organization (instrument technician, plant operator, ICT partner, etc) and descriptions of tasks these different roles carry out.

In order to perform this kind of research, CAIRIS tool has been used. CAIRIS, indeed, allows a detailed characterization of personas, specifying their roles, their motivations, to perform their tasks, and their capabilities. Furthermore, CAIRIS allows the identification of environments, representing different possible working scenarios, helping in this way the team to separate concerns and develop, in this way, more specific models.

Thanks to CAIRIS, it has been possible to create a complete task model, able to link with each other all the elicited informations. In particular, each persona has been assigned to a role and to tasks he performs, while tasks have been associated with assets of concern in the given environments. ACME Water has also a security team which has been designated to create a risk database over which rely, in order to track incidents and risks associated with the ability to endanger the water delivery from this critical national infrastructure. In order to accomplish this task, they exploited again the functionalities offered by CAIRIS, which has helped them to create a risk model, tailored to the requirements elicited in the task model, carried out previously by the UX team. Thanks to the features offered by the tool, each risk is associated with a collection of assets, either threatened by an attacker, or exposed as a vulnerability that an attacker might exploit. For each asset the security properties to protect are identified, for each vulnerability, moreover, the security properties it endangers are specified. By linking vulnerabilities with assets, risks can be elicited

and a score for those risks is automatically computed.

Among the several personas and workers there is Barry [9]. Barry is an authorized instrument technician of ACME Water and much of his work involves equipment modification and instrument calibration; this work arises from specific requests from his superiors, or as activities which are part of on-going projects to improve plant efficiency. As part of these changes, Barry may need to modify alarms on outstations, and make minor changes to PLCs and HMIs.

By using the SCADA system, it is possible for Barry to remotely configure SCADA Human Machine Interface (HMI) files, which would otherwise only be possible by being present in water treatment plants, using the HMI.

Each operation which involves the access to resources, which reside within workstations connected to the internal network of the company, such as modifications of files, preview of confidential documents, execution of software processes, has to be evaluated from an access control policy. The security team, after some years of adoption of a role-based access control mechanism, has decided to pass to a more dynamic solution, which involves the evaluation of risk parameters when dealing with access requests. In order to rely on an approach of this kind, the SAFAX service has been chosen. In particular, they decided to rely on the complete XACML infrastructure offered by SAFAX, thus delegating to it each functionalities needed during the policy evaluation. This means that whenever a user tries to perform an access over a resource, the SAFAX PEP is contacted and it is the one to initially block the user's access, until a valid authorization response is communicated to it by the PDP.

The idea of adopting this new solution, when taking authorization decisions, came from the observation that, in the last years, due to a significant expansion of the water treatment plants, the dynamics and the organization of the work, within the company, have changed: it has been observed, for example, that the number of accesses to the SCADA workstations from within the plants has significantly decreased, highlighting how the remote access, to these workstations, has become the preferable way for carrying out most of the tasks. From this perspective, the two environments, elicited during the characterization of the models, are precisely *insite* and *offsite*, which separate the operations that can be performed being within a water treatment plant from the ones that can be performed from the outside.

Starting from these two scenarios, a whole new set of threats, now, has to be accounted for, since the access to the resources may be performed from potentially insecure external machines, thus, the fact of possessing the authorization permissions coming from the intrinsic trust put in a role is not anymore sufficient to guarantee an appropriate level of protection for the system.

### **6.1.1 Situation 1:**

The ACME Water security team is made aware of a new piece of Windows malware that, potentially, has affected every laptop used by instrument technicians at ACME Water. The malware tampers with SCADA HMI files, and the malicious execution is invoked each time the laptop connects to the public Internet. This malware might affect Barry's laptop when remotely connecting to the SCADA system offsite, but does not affect him if he alters the file using the laptop on a treatment plant site private wi-fi network or on a workstation at the plant.

ACME Water's security team, after having built up the risk model and eliciting, among the different identified threats, all the potential risks that could result either from performing an access from within the plant's internal network or from outside, wants a policy that denies the access to SCADA HMI files under the following condition: the risk posed by any threat exceeds a threshold of 6, where the risk level depends on whether the SCADA HMI file is accessed insite or offsite. In order to have an implementation of this kind, the security team has uploaded, on SAFAX, a specific policy which has as target's subject *SCADA HMI files*, as target's environment *offsite* and as target's actions *read* and *modify*. Furthermore, in order to exploit the risk service,

they specified as UDF function the one linked to the risk API that, given an asset, identified among a set of 50, and an environment specified in a request, is able to retrieve the highest risk score associated to that asset in the given environment, among all the possible threats (table 5.3), which are roughly 50 for each environment.

Given the likelihood with which has spread in systems similar to ACME Water and the severity of the vulnerability that is associated to this threat, which mainly would endanger the confidentiality of informations and the integrity of the system, the highest risk retrieved is precisely the one posed by the new Windows malware and is rated as a 9, in case of a remote access, and as a 4 if, instead, the access to the file happens from a workstation within the infrastructure's private network domain. Consequently, if Barry attempts to access the SCADA HMI file using his laptop offsite, his request should be denied.

### 6.1.2 Situation 2:

One of the advantages that CAIRIS brought within the company, by adopting it, has been the fact of offering itself not only as a security, but also as a usability requirements management tool, allowing the security team to associate personas and tasks to assets, tracking the usability implications of these tasks. Indeed, it could happen that if assets were compromised or made unavailable, the workload of the associated user would increase to the extent that he might violate security in order to complete the task.

In order to cope with this possible issue, the ACME Water security team wishes to apply an additional policy, in SAFAX, such that if the denying someone access to a resource would otherwise affect a critical task, access should be granted if the associated risk scores less than 8. Barry is often called out to troubleshoot problems at sites and kiosks within his area of operations, which can involve software changes, ranging from simply downloading software to a device to clean up its memory, through to re-generating the software from configuration sheets if no software backup is available. In such a situation and keeping in consideration the warnings made by the security team, which follow from the situation, in 6.1.1, previously presented, regarding the new malware discovered, Barry needs to modify a SCADA HMI file while simultaneously carrying out another job offsite (an emergency requiring the technician to fix multiple issues across a wide geographical area). If the access to the file is denied then Barry's behaviour becomes undefined and he may attempt to carry out the task via some undocumented means: for example, he might drive to the affected plant to modify the SCADA settings locally, thus increasing the response time for the other emergencies, or he might use some questionable workaround, skipping steps in the task that may lead to a security violation.

This possible issue has been already foreseen by the UX team, thanks to the deep research carried out, thus the CAIRIS service has been configured in such a way to mitigate the risk score by decreasing it of 2 points, if the system is made aware of the other pending emergencies assigned to the same technician. It will be up to the authorization service, in particular to the *CAIRIS module*, offered by SAFAX, to check if the final mitigated risk can be met by the threshold specified in the policy.

In such a situation, the risk related to the access to the resource will be now rated as a 7, consequently, Barry should be allowed to access the resource.

## 6.2 Scenario 2: Healthcare facility

Healthcare organizations are complex and dynamic. Within an healthcare facility there is a huge number of tasks to be performed and a huge number of people working in it, everyone fitting in his own role. It is usually present a management section in order to support and coordinate the

services that are provided within the organization, an administrative section to take care of all the bureaucracy and human resources, then there are doctors, nurses and possibly also students which are doing their internship within the hospital. Furthermore, cannot be forget the staff that works inside the facility like cooks, laundry workers, cleaners, security personnel and information system administrators. All those figures and roles must be taken into consideration when developing a scenario of usage, in particular if they have access to the facility's network, since everyone of them could represent a possible threat.

The intrinsic nature of this kind of facilities makes these scenarios highly dynamic and difficult to predict a priori, the same difficulties stand in the definition of authorization policies that allow to maintain an high level of security but, on the other hand, that allow an acceptable degree of freedom for users, that interact with the information system of the facility, in order to perform operations that can be particularly sensitive and urgent.

The adoption of a dynamic context aware risk-based access control mechanism would help the system to take more thoughtful authorization decisions, which may vary depending on the environments the system is facing.

In this kind of scenario, the security policy adopted by the organization foresees two kinds of environments: *critical* and *not-critical*. The critical one represents a red code emergency situation with high urgency requested, the not-critical one, instead, represents green or yellow code emergency, where the situation is easier to manage and there is not such urgency. The two environments are modelled with CAIRIS, assets of concerns are grouped in three main categories: *informations*, which are mainly characterized by integrity, confidentiality and availability properties; *hardware*, which represents the physical hardware devices used inside the facility, mainly characterized by availability properties; *software*, which represents programs and tools used by the facility personnel, characterized by integrity, availability, and accountability properties. Patients' confidential records probably represent the most sensitive assets within the information class, due to the high confidentiality that characterizes them.

An external security firm has been entrusted with the development of risk models contextualized to the different identified environments, in such a way to create a database of risks to be exploited during policy evaluations.

By analysing the developed model, built up on the facility scenario, with CAIRIS, one of the main threats identified by the security team is the intentional or unintentional disclosure of sensitive informations, regarding patients, to the unauthorized personnel. For this reason, some facility's executive manager, involved as a stakeholder, during the elicitation requirements and risks definition process, is more prone to use a policy which implements very strong rules regulating informations' accesses; at the same time, though, it has been observed, by past experiences, that a too conservative policy has sometimes resulted in the arising of unacceptable issues, that prevented some patients to be appropriately taken care of, putting at serious risk their physical wellness.

With the identification of separate environments, it has been possible to give a finer-grained resolution to the problem. Authorization decisions will be taken evaluating the risk of a request against a threshold and depending on the environment.

The information system adopted by the facility, in order to manage authorization requests, relies on an application which autonomously implements the PEP, PIP and CH components, characterizing the XACML-based architectural framework, but the CH has been implemented in such a way to contact the PDPs configured on SAFAX service, which represents the authorization provider and the trusted repository point where to deploy policies. It will be up to SAFAX service, then, to retrieve the risk scores from the CAIRIS database, configured by the security team. An implementation of this kind exactly represent the Distributed and Delegated Trust model (figure 5.2). According to the deployed policy, if the risk exceeds the value of the threshold the permission is denied, otherwise it is granted.

A first check performed by the policy is to identify the role of the subject, which is associated to

the subject credentials as an attribute and is provided by the PIP; a second check foresees to verify whether the request comes from an authorized terminal or not. Indeed, sensitive informations, such as patients' medical records can be accessed only from workstations situated in the same ward where the patient is treated. The entity which is responsible of identifying the terminal is again the PIP, which saves the information as an additional attribute to be used in the policies. A third check is, moreover, performed over the kind of access the user is trying to perform (read, update, etc).

In order to retrieve the risk, the request should be filled with the following parameters: the resource that the subject is trying to have access to, the threat associated to the kind of access it is requested and the environment in which the authorization request is submitted. The specification of these three parameters allows to use a customized UDF, inside the reference policy, which exploits the API able to retrieve the highest risk score for an asset, given a certain threat, within the specified environment (table 5.6). In such a scenario, 120 sensitive assets have been identified by the security team during requirements elicitation, which may possibly be endangered in both the considered environments, each one attackable by roughly 50 threats.

### **6.2.1 Situation:**

An emergency happens, a patient in life-danger conditions arrives and no doctor is at the moment available, due to the high workload given by multiple emergencies. Only nurses, which have a lower security level in respect to doctors, are able to receive the patient and they need to look into his clinical records in order to find whether a certain treatment could be good for the patient or could be harmful. By accessing his clinical records, confidential and sensitive informations about the patient would be disclosed.

Normally, which means in presence of a not-critical emergency, or during the normal operations performed, without a specific association to an emergency, due to the security policy adopted by the facility, nurses wouldn't be able to access that kind of resources, since the risk threshold set to 6 by the risk policy, developed by the security team, would block the access. For example, a reading mode request, made by a nurse using a terminal, within the same area where the patient is treated, will result in a score of 8, that if not mitigated in some way, would result in a risk too high.

In this specific situation, though, the red code emergency would set the environment to critical: the highest risk score associated to a reading access to the patient's clinical records, considering the possible information disclosure threat, identified within the critical environment, will be retrieved. In such critical scenario, the risk posed by the threat has been configured to assume less dangerous values, by relaxing, in CAIRIS, the relevance value of the associated security properties, giving as a result 6, the minimum acceptable value in order to grant the permission, and allowing in this way the nurses to access the patient information records and accomplish their task.

## **6.3 Scenario 3: NeuroGrid**

NeuroGrid [54] was a UK Medical Research Council funded project to develop a grid-based collaborative research environment for different clinical researcher communities and a case study based on this scenario is going to be presented now.

The main goals of this collaborative system are collecting, analysing, interpreting and secure archiving of neuroimaging data. According to the NeuroGrid CAIRIS example [53], three environments are defined: Core technology, characterized by NeuroGrid infrastructure operations; Psychosis, where the focus is put on the integration of serial MRI scans and behavioural data, and on the development of methodologies to perform analysis over those data, together with the

development of a general ontology for psychosis related data; and finally Stroke, which focus on improving the infrastructure that allows the sharing, among different researches centre, of large archives of images linked to key metadata for diseases which require long term study. The sensitivity of those clinical data and their distributed nature drives the need to find secure and effective ways of accessing and managing them.

Since the community of researchers, which decided to adhere to the project, are spread all over the world and each one of them relies on a specific repository for accessing and managing resources, there is the absolute need for coordinating the access to those resources in such a way to protect the informations but at the same time to guarantee an appropriate level of usage. To this purpose, SAFAX has been adopted as a unique point were to deploy policies and each repository interface used among the different communities has been registered within the SAFAX service repository, in such a way to coordinate and group all the authorization policies in a single place, and allowing to specify them once and for all and not anymore singularly for each repository's authorization system, incurring in the risk of possible misalignment among the different communities.

Basically, a Distributed and Delegated Trust model is implemented, with the repository's specific authorization systems implementing their own web interfaces relying on specific PEP, PIP and CH components. In order to rely on the same policies, each implemented CH will contact the dedicated PDP, configured on the SAFAX server and, if requested by the policy, the *CAIRIS module*, again implemented on the SAFAX server will be invoked in order to evaluate a risk request and retrieving this risk from CAIRIS server.

A preliminary configuration phase is indeed necessary both in SAFAX to configure PDPs, deploy policies and configure the credentials that those PDPs need in order to contact the CAIRIS service, and, in CAIRIS, to develop a risk model, able to elicit and quantify the possible risks arising from the identified scenarios, in such a way to create a complete database of risk notions.

One of the main issues, characterizing the management of resources in this collaborative network, stands in the fact that when interrogating the NeuroGrid databases, it is very simple to aggregate and disaggregate informations, forcing the system to show all the wished results, with the risk of releasing too explicit informations that could compromise the confidentiality of data and clearly the privacy of people. For this reason, anonymization's measures have to be adopted.

In order to cope with these security requirements, which require a certain dynamic and responsive ways of managing informations and involving outcomes which cannot be easily predicted, the authorization system has been configured to rely on policies which consider risk scores, which varies with the environments and the kind of request is performed.

In particular, a risk is evaluated depending on the number of subjects involved in the research and whether the access to the database happens from a machine inside or outside the centre. Researchers, anyway, are strongly discouraged to adopt this last option and whenever they need to access data being offsite they are obliged to submit a report justifying the kind of work they are carrying on. The thresholds instead are set only by looking at the number of subjects the results are based on; in this way the fact of being offsite will just make the risk worse, thus increasing the probability of rejecting the request.

In particular, policies have been defined in such a way that if, in any query performed, the results obtained are based on less than 10 subjects, the risk threshold is set to 1, meaning that in any case the risk of accessing those kinds of informations is too high, since it could bring to the identification of specific subjects and none of the people should be allowed to access those sensitive data. When the number of subjects is higher than 10 up to 20, then the risk's threshold is set to 4. When, instead, the number of subjects is higher than 20, then the threshold is set to 7. In this kind of scenario, CAIRIS has been configured in such a way that assets are represented by diverse aggregations of sensitive informations and environments are represented by the aggregation of a particular research scenario (Core Technology, Psychosis and Stroke) with the fact of accessing the informations either from a machine connected to the internal network of a research centre or from the outside.

Policies, in SAFAX, have been configured to retrieve, through the proper UDF, the highest risk score for a resource in a specific environment.

### **6.3.1 Situation:**

A community of researchers is particularly focused on the Stroke environment, investigating and analysing data and images concerning some special kinds of diseases. The work of the researchers is now focused in comparing different cases of the same disease with the goal of finding some common ground in order to make some guesses about distinctive characteristics in people that probably are the main cause in the arising of the disease. More specifically, in this kind of scenario, they have to deal with a database of 120 subjects, which are the assets considered at risks. In order to come up with the first conclusions, a lot of studies, analysis, data processing and queries towards NeuroGrid database have to be done.

Claire is a clinical researcher and she is responsible for this researchers' community. She authorizes requests that other members of her group make for accessing NeuroGrid, and is officially responsible for making sure her colleagues aren't misusing their access. Anyway, sometimes, in order to speed up some works and simplify some bureaucracy's issues, she's used to let Maria and Mark, two students doing their internship in the research centre, use her digital certificate, in order for them to have a broader access to the NeuroGrid resources and perform more accurate analysis. The two young students have been instructed about the security procedures adopted by the centre, anyway the perception they have about security is quite superficial, especially if it is related to cyber-threats.

Mark, which is really enthusiastic about the project is working on, in order to complete some analysis he was doing during the day, access the NeuroGrid database, with Claire's certificate, from his laptop at home and starts interrogating it. Unfortunately, he is not aware of the fact that a malicious software is acting in background and downloading any web page he has access to. Since the requests he makes are quite specific (he wants to retrieve the date of birth, the place of birth and the nationality of the parents of all people affected by the disease under study that currently live in the south-east of England) and they could be possibly dangerous, from the confidentiality point of view (it happens that the data refer to just 15 subjects), the risk score will be evaluated as an 8, a value that takes into account the confidentiality of the data plus the offsite access mode. The risk score is greater than 4, so he won't be able to successfully obtain the wished results, avoiding in this way a possible undesired disclosure of informations, that would have been captured by the malware.

After having realized that such informations are not possible to retrieve, at least not from being outside the centre, he decides, for the moment, to settle for a more general research (30 subjects involved), that brings to a risk score of 6, now to be compared with a threshold of 7, letting in this way Mark access the informations. At this point, the results obtained are sufficiently aggregated in such a way to loose their confidentiality property, so even if the malware captures those informations they are not considered a threat.

## Chapter 7

# Performance Results

In this chapter the results, obtained after having carried out a performance analysis of the developed system, are going to be presented and discussed.

The analysis has been performed by deploying the CAIRIS' and SAFAX's components into three different virtual machines on the same host machine, by using the VirtualBox virtualization environment, in such a way to emulate the Distributed and Delegated Trust Model presented in [40] and already shown in figure 5.2. In particular, the PEP, PIP and CH web service components have been deployed on one machine; the PDP, PAP, CAIRIS module and a the database, where to store policies, CAIRIS' credentials and models informations on a second machine; finally, the CAIRIS service, together with a database where to upload the CAIRIS models with which to interact, on a third machine. The three virtual machines have been configured to be on the same internal network in order to communicate. Figure 7.1 shows the adopted deployment diagram.

From the figure, VM1 and VM2 represent two 64 bit Linux Ubuntu 14.04 LTS virtual machines, while VM3 is a 64 bit Linux Ubuntu 16.04 LTS virtual machine. The latter exploits the currently last Linux operating system version, which is the suggested and most tested one, in such a way to correctly install the tool. On VM1 and VM2 the web services run on an Apache Tomcat 7.0.76 server and on VM2 a MySQL 5.5 Database has been installed. On VM3, instead, the CAIRIS server runs on an Apache2 HTTP server and interacts with a MySQL 5.7 Database.

The performance analysis carried out in this chapter has as object the policy evaluation time for a request, which needs a risk score retrieval in order to be evaluated. By performing an analysis of this kind, it has been possible to observe the average policy evaluation times at different level of granularity, understanding for each component the average percentage of time it spends in order to accomplish its tasks. Furthermore, a specific focus has been maintained over the risk retrieval time from CAIRIS, observing how it varies when the load, in terms of number of risks stored, of the CAIRIS database changes and when the expressiveness of the policy changes as well, by exploiting the four APIs supported and described in detail in Chapter 5.

For each experiment, the average values obtained as results have been computed over 50 runs, in order to make a sufficient number of tests smoothing out the variance of the results.

A first set of results are shown in figure 7.2. These results have been obtained exploiting the usage of API 1 (table 5.3) and varying the number of assets and the number of threats present in the CAIRIS database and assigned to a specific asset. The API should retrieve the highest risk score among all the possible threats assigned to the specified asset, thus we expect that an higher number of threats, for an asset, would result in an higher number of operations to perform, in order to compute the maximum among all the related scores. Figure 7.2 shows exactly this behaviour in the data, in particular we can observe that, if we fix the number of assets present in the database, by halving the largest number of threats, associated to an asset and taken into consideration in this analysis (100), the risk retrieval time is approximatively halved too. By further reducing the number of threats from 50 to 10, instead, the risk retrieval time is reduced of the 58%, 55%, 65%,

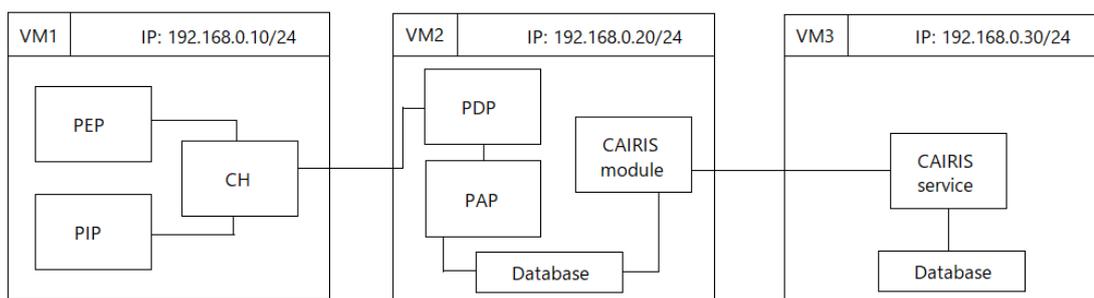


Figure 7.1. Deployment diagram for CAIRIS' and SAFAX's components in three different virtual machines.

67% respectively for the cases with 120, 50, 10 and 1 assets, thus showing a non linear decreasing behaviour, in respect to the first variation. The situation, where only one threat is associated to the asset, whose we want to retrieve the risk, reflects the case with the minimum necessary time needed to retrieve a risk score once the specified asset has been identified in the database.

Furthermore, it is also possible to observe how the times vary when the number of assets present in the database changes; indeed, the database needs some time in order to be able to identify the specified asset, passed in the API as parameter, among all the assets present and stored in the database. In order to identify an entry into the database, MySQL exploits the B-tree data structure, which is a generalization of the binary search tree, letting a node the possibility to have more than two children. The great advantage of using an approach of this kind is that sorting is allowed and the look-up can be done in a logarithmic number of steps and speeded up with the usage of indexing [52].

While varying the number of threats bring us to more significant time variations, in the order of hundreds of milliseconds, if we fix a certain threats' load, the variation of the number of assets bring us to a less evident variation of time, represented by tens of milliseconds, not changing significantly the order of magnitude. This bring us to an important consideration, which can make us to state that the most demanding operations are the ones concerning the comparisons of risk scores and not the ones involving the search of a resource in the database.

The worst case scenario is clearly represented by the situation in which 120 assets are present in the database and 100 threats are associated to the asset whose we want to retrieve the risk. By decreasing either the number of threats or the number of assets or both, the risk retrieval time decreases as well, until reaching it's minimum possible value represented by the situation with 1 asset stored in the database and just 1 threat associated to it. Figure 7.3 shows, for the four realizations obtained by loading the database respectively with 120, 50, 10 and 1 assets, the decreasing behaviour just described: the figure shows exactly this lack of linearity in the decreasing trend, and roughly tends to find a point of convergence, for each realization, in the scenario with just 1 threat assigned to an asset.

Since the variation of the risk retrieval time, when the number of threats per asset varies, is a perceptible and clear understanding, given from the fact that the more the risk scores in the database and the more have to be the operations in order to compute the maximum among them, it has been decided to carry on a further analysis, able to focus on the differences in accessing the risk informations by varying the number of assets present in the database, for each one of the four APIs, described in Chapter 5, keeping the number of threats per asset constant. In particular, the number of threats assigned to an asset has been fixed to 100, equally subdivided into two environments (50 threats per environment).

Figure 7.4 shows the results obtained for each API at the variation of the number of assets populating the database of risk's informations. It is possible to observe how the trend, which

API 1				
times in ms	100 threats	50 threats	10 threats	1 threat
120 assets	755.88	414.02	172.408	62.44
50 assets	719.96	370.02	168.64	52.9
10 assets	655.7	351.24	121.68	41.42
1 asset	529.78	269.6	85.14	28.26

Figure 7.2. Average risk retrieval times in ms, obtained for different loads, in terms of number of assets and threats, of the CAIRIS database, exploiting API 1.

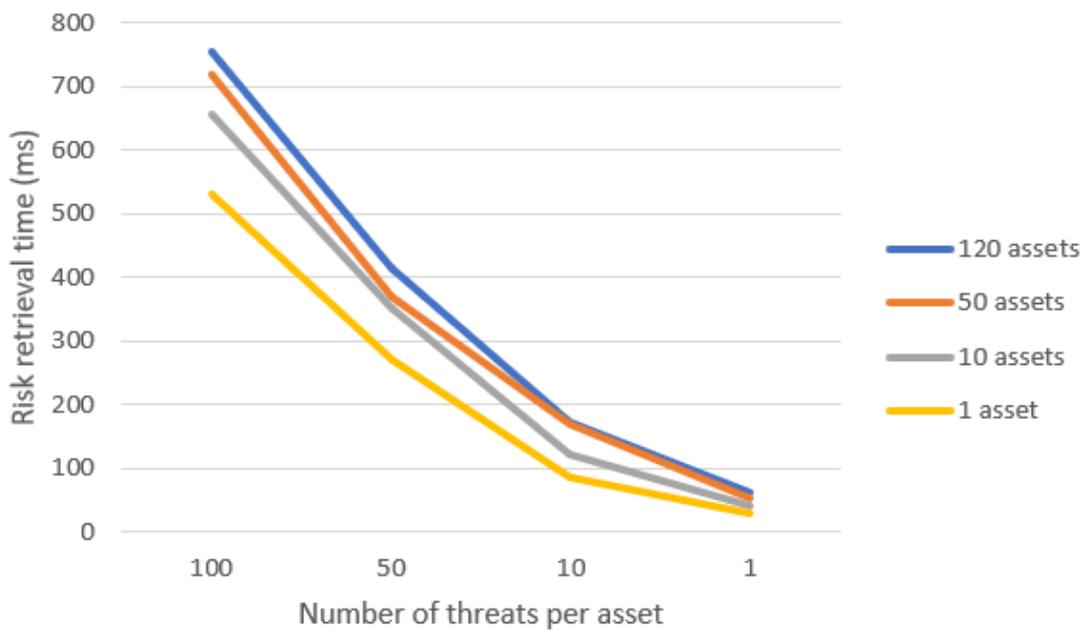


Figure 7.3. Behavioural trend obtained from the results in figure 7.2.

involves shorter times for smaller loads of assets, is again confirmed for all the APIs.

Moreover, it is interesting to compare the risk retrieval times obtained by using the four defined APIs. In particular, as aforementioned, the API 1 (table 5.3), allows us to retrieve the highest risk score, searching among all the possible threats and environments, given a certain asset, thus representing the heaviest task to perform, requiring to compare all the 100 risk scores before having the maximum.

API 2 (table 5.4), instead, allows to retrieve the highest risk score for an asset, given a threat type, among all the possible environments. Since, in the experiments performed, the number of environments has been fixed to two, once the asset and the threat have been identified, the comparison to make is just one, in order to decide which risk, among the two, is the highest. Since the most demanding operations are reduced to the minimum, it is possible to obtain, as a result, very short times, which become even shorter reducing the number of assets present in the database. In particular, the case with 120 assets matches exactly situation 6.2 described in

	50 threats per environment			
times in ms	API 1	API 2	API 3	API 4
120 assets	755.88	81.4	385.417	33.0385
50 assets	719.96	74.22	316.94	31.78
10 assets	655.7	35.12	296.74	27.06

Figure 7.4. Average risk retrieval times in ms, obtained by fixing the number of threats per asset in the database and varying the number of assets, exploiting the four APIs.

Chapter 6, thus in such scenario, the authorization process will need 81.4 ms, on average, in order to retrieve the correct risk score.

By proceeding with API 3 (table 5.5), the highest risk for an asset, among all the possible threats in a specific environment, can be retrieved; in such a case, once the asset and the environment of interest are identified, the risk scores to compare are exactly 50 (remember that there have been defined 50 threats for each of the two environments, for a total of 100 threats). For this reason we expect to obtain better performances than the ones obtained from the usage of API 1, but worse in respect to the ones obtained with API 2 and the data confirm this forecast. This kind of scenario can be, actually, compared, from the operational point of view, with the one presented before, in figure 7.2, and involving the API 1 where 50 threats have been assigned to an asset; the results obtained, indeed, are quite similar, surely in the same order of magnitude. The differences among the two categories of results, represented by tens of milliseconds, can be probably addressed to a little variance of the results characterizing each realization when making an experiment.

In Chapter 6, scenarios 6.1 and 6.3 have been thought to exploit exactly API 3, in the case with 50 and 120 assets respectively, thus needing approximately 316.94 ms and 385.417 ms, respectively, on average, to retrieve the risk scores.

Finally, API 4 (table 5.6) allows to retrieve the highest risk score for an asset, given a threat, in a specific environment, thus restricting, even more than API 2, the range of possible risks among which make the comparisons. In particular, by using this kind of API, no comparisons are needed, since the informations specified in the API point us to a punctual and singular risk score. By eliminating the comparisons, which represent the most demanding operations, we can obtain the best results observed for the risk retrieval time, in the different APIs: 33 ms in the case with 120 assets, around 32 ms in the case with 50 assets and 27 ms in the case with 10 assets.

The comparison among the different APIs allows us to understand the performance implications resulting from the different possibility of expressiveness for a risk policy. What emerges is that well-defined scenarios, where threats and environments can be easily identified, or where we consciously want to be more precise when retrieving a risk, by specifying the exact threat and eventually also the environment involved, benefit from the best performances, while when the risk retrieval becomes a wider and probably more conservative process, that wants to consider a larger spectrum of possible risks, the performances are affected.

Until here, the focus has been kept over the only parameter which may vary during policy evaluation, due to the workload of the system, and we saw which are the possible reasons characterizing its variations. Let's now see the other parameters characterizing a complete policy evaluation process, which should remain constant, since they do not depend on any other factor than possible fluctuations given from the workload of the server or of the network if an HTTP call is involved. These parameters are:

- *PAP working time*: the average time needed to retrieve a policy from the PAP.

- *CAIRIS authentication time*: the average time needed by SAFAX to retrieve the CAIRIS credentials and use them to authenticate on CAIRIS, obtaining a valid session.
- *CAIRIS open DB time*: the average time needed by SAFAX to retrieve the desired CAIRIS database associated to the selected credentials and open it on CAIRIS.
- *Risk Evaluation time*: the average time needed by the CAIRIS module, within SAFAX, to evaluate the risk against a threshold.
- *CAIRIS module working time*:  $(\text{CAIRIS authentication time}) + (\text{CAIRIS open DB time}) + (\text{Risk Evaluation time}) + (\text{spare time to perform the necessary operations which guarantee the correct functioning of the module})$ .
- *PDP working time*:  $(\text{PDP average time needed to evaluate a policy}) - (\text{PAP working time}) - (\text{CAIRIS module working time})$ .
- *CH working time*:  $(\text{CH average working time}) - (\text{PDP working time})$ .
- *PEP working time*:  $(\text{PEP average working time}) - (\text{CH working time})$ .

PDP, CH and PEP working times have been described with a subtraction, since the times have been computed by taking as initial value the instant in which the HTTP call of the API contacting the module is performed and as final value the instant in which the API returns with an HTTP response to the entity which called it. It follows that in this way the PDP working time would include in its operations the PAP and the CAIRIS module working times, the CH would include the PDP working time and the PEP would include the CH working time in a sort of hierarchical manner. By subtracting the inner working time of the “son” modules, it is possible to obtain the effective working time for the “father” module.

The results, that have been obtained from the experiments described above, showed that while the procedure carried out by SAFAX to contact the CAIRIS service, in order to open the desired database, is quite fast, around 40 milliseconds on average, the authentication procedure is much heavier, oscillating from 950 ms to 1100 ms, due to the different experiments’ realizations and representing, in this way, one third of the total average policy evaluation time.

The average risk evaluation time, instead, is around 65 ms and the average policy retrieval time from the PAP requires about 160 ms.

By summing up all the values until here obtained, it is possible to understand that, by using API 1, the average CAIRIS module working time requires around 2200 ms when 100 threats are present in the CAIRIS database, decreasing to 1900 ms when the number of threats is 50 and further decreasing to 1700 ms, when only 10 threats are present, and to 1600 when just one threat is present.

Focusing on the scenario with 100 threats present in the database, by using API 2, the CAIRIS module average working time is of 1600 ms, while, by using API 3 is of 1800 ms and by using API 4 of 1500 ms. The variability in those times is clearly due to the fact that the different APIs, as described above, allows to access the data with different levels of granularity. It follows that the great majority of the operational time, for the module, is spent performing authentication, which is the heaviest task.

Actually, the CAIRIS module average working time represents the main time component within the whole policy evaluation process; we need then to add 500 ms, on average, of PDP evaluation time of the policy, 290 ms of CH operations, which involve: retrieving the IP address of the configured PDP and constructing the URL with the necessary parameters to be passed to the PDP. Furthermore, 320 ms of PEP operations have to be accounted: retrieving the IP address of the configured CH, checking the validity of the XML format for the request and constructing the URL with the necessary parameters to be passed to the CH. It is important to notice that in

the performed experiments, no additional attributes from the PIP have been added to the policy evaluation process, thus in this case the CH just vehicles the request, brought by the PEP, to the PDP.

As a result, the usage of API 1 allows to obtain an average policy evaluation time of about 3370 ms and API 2 allows to obtain around 2270 ms, which represents the time needed to evaluate a risk policy in scenario 6.2. API 3 let us to obtain as result 2970 ms, thus quantifying the risk policy evaluation time needed in scenario 6.3 with 120 assets, while we would remain in the same order of magnitude, just some tens of ms less, if we consider scenario 6.1, where the assets are 50. Finally, by using API 4 we can obtain around 2670 ms.

## Chapter 8

# Conclusions

In this chapter, we draw conclusions for the thesis work that has been carried out, summarizing the main features of the results obtained. At first, we try to make a critical thinking of the work done, trying to recall the steps that had to be faced in order to obtain the final realization of the system. Lessons learnt during the process and open issues are outlined. Moreover, future works, that could be possibly undertaken, in order to be integrated in the developed system, are proposed.

### 8.1 Lessons learnt and open issues

The purpose of this thesis work has been to implement a context-aware risk-based authorization system, by extending the SAFAX tool and integrating it with a second tool, CAIRIS. SAFAX proposes itself as a framework for XACML policy evaluation, while CAIRIS is a security and usability requirements management tool, that allows the creation of risk models and the categorization, the specification and the storage of risk's related informations regarding assets, threats, vulnerabilities and environments.

A first goal of such a framework is to enhance the well-known role-based paradigm, considered too static and not sufficiently responsive to cope with highly dynamic scenarios where it is difficult to predict which authorization permissions should be granted to a user, in order to protect the system from possible misbehaviours, and at the same time allowing a sufficient usability degree in order to let the users accomplish their tasks.

Some usage scenarios, which would benefit by using the developed system, have been presented in Chapter 6. These scenarios show, indeed, how the dynamism of this system would help to solve some security implications coming from improper authorization decisions.

A second goal of the framework is to let the system be context-aware, meaning that it should recognize the environment in which is working, in such a way to adapt the authorization decisions accordingly. In this way it is possible to take finer grained decisions which help to further enhance the dynamism of the authorization process.

Although SAFAX has been built in such a way to be easily integrated with external services, by exploiting User Defined Functions, which represent customized ways of treating attributes within XACML policies, several issues arose when came up the idea of integrating it with CAIRIS tool. The main problem was that CAIRIS uses some form of authentication which wasn't supported by SAFAX and it was not possible to directly contact the CAIRIS' APIs, in order to retrieve the risk values associated to the resources and to be used during the policy evaluation procedure.

Before proceeding with the risk's retrieval part, the first step performed for the implementation of the system has been to understand the authentication mechanisms supported by CAIRIS. Two mechanisms are currently supported: the first, very trivial, involves the usage of a static identifier,

recognized by CAIRIS, which can be simply included in the URL of the API as a data parameter. The adoption of this method only allows to exploit the CAIRIS service with a default test account, so it clearly has a very limited usage, which is just for testing purposes. The second option instead allows to rely on the basic HTTP authentication mechanism, which is much better because it allows each user to contact his own CAIRIS model, developed within his personal account; for this reason it is more suitable to be adopted in a production environment.

The CAIRIS credentials and database informations, that should be used to successfully carry out the authentication with the CAIRIS server, have to be specified during a preliminary configuration phase on the SAFAX tool, through the offered GUI. In particular each demo, present in SAFAX, will have its own credentials specified, storing these informations in the SAFAX database. This kind of implementation, concerning the management of credentials, has been necessary in order to support the autonomy of the system when performing the authentication procedure, which can be done automatically without requiring a third party software, which exploits the SAFAX service programmatically, to modify the currently supported methods in order to pass the credentials when sending an authorization request.

This means that any application, which develops a set of methods able to contact the appropriate SAFAX PEP APIs and that conform to the requirements of these APIs, would be able to exploit the authorization service provided by SAFAX.

The weakness of such approach is represented by the fact that confidential informations, belonging to CAIRIS, have to be managed and stored not only within the CAIRIS domain, but also in the SAFAX domain too. This means that, from a security point of view, those credentials could be attackable in two different points, thus increasing the chances for a possible attacker, that wants to steal those credentials, to find exploitable vulnerabilities in the two systems. The alternative would be to modify the PEP's APIs in such a way to support a remote passage of credentials between the third party application and the SAFAX service. By adopting a solution of this kind, there wouldn't be the need for the management of CAIRIS credentials in the database but, on the other end, those credentials would be susceptible of attacks, by a MITM, when passed to SAFAX. Once the authentication part has been successfully completed, SAFAX will include the identifier of the negotiated session as a data parameter in the URL API, in such a way to prove his authorization to make the specific risk request.

Beside the identifier of the session, the necessary URL parameters have to be included in the APIs: resources, threats and environments are the requested informations in order to retrieve a risk score. These values are extracted from the XACML request, submitted by a user, by the PDP which contacts the developed *CAIRIS module*, internal to SAFAX, by means of a local API, and integrate in it the extracted parameters.

The *CAIRIS module* has been implemented as a RESTful web service, where the information about the SAFAX ongoing session is passed in the API through which the module is contacted. The module is in charge of adopting one of the supported authentication mechanisms, just described, opening the desired CAIRIS database, with which interacts, and constructing the HTTP URL to use when retrieving risk. Then, it implements its own risk evaluation method. The standard method that has been chosen is threshold based, meaning that, in order to take the final decision, the risk value retrieved will be compared to the threshold specified in the policy.

The idea of integrating the risk threshold directly within the policy comes from the fact that it represents a parameter strictly connected to the resource or to a precise group of resources. Risk is a concept which is directly linked with an asset as well, for example we can say that each asset has its own confidentiality value, thus the risk impact, that a possible unauthorized access could have over an asset, strictly depends on the confidentiality property of the asset itself. For this reason, coherently, a risk threshold should necessarily be referred to the specific asset too and could not be defined as an absolute parameter valid for an entire scenario.

Moreover, a risk, which refers to a specific asset, changes if we consider the asset in slightly different contexts, indeed, the risk retrieval process varies according to the kind of policy we are

considering: there are policies which consider some aggregated form of the risk value retrieved, others which consider more specific risk values, by further specifying the threat associated to an endangered asset, others, furthermore, support a much finer grained definition of the risk, by allowing not only the specification of the threat, but also that of the environment to consider when retrieving the risk associated with an asset. It follows that the risk threshold should be adapted accordingly too.

It is true that defining properly a value for the threshold is not an easy task, for this reason, probably, it is not sufficient to just carry out a risk analysis but it would be appropriate to perform some kind of sensitivity analysis too, in order to appropriately set risk's thresholds as well, as a preliminary phase, after having built up a risk model on CAIRIS. Sensitivity analysis indeed has been proposed as “the study of how the uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input”[63]. Such kind of analysis could be of help in order to test the robustness of the threshold based decision mechanism implemented, as well as the goodness in choosing the input parameters with a certain degree of uncertainty and variability. In [11], a sensitivity analysis in view of a threshold decision making process is presented and applied to a real-life dynamic scenario in the healthcare field, to test the robustness of the decision that a clinician should take about whether or not to start a certain treatment to a patient, basing on the probability for that patient to find himself in a particular health condition. In the example proposed, a sensitivity function representing a threshold probability is built and the results observed at the varying of a parameter  $\theta$  of the model, with the goal of identify the region within which a certain decision shouldn't change. A similar approach could be eventually used in our case, to the same purposes, by selecting the CAIRIS risk rating function ([28]) as the sensitivity function and by observing its behaviour at the varying of the two parameters which characterize that function: the threat likelihood and the vulnerability severity. We may take those two parameters into consideration as input parameters, to our problem. They both represent uncertain quantities “due to incompleteness of data and partial knowledge of the domain under study”, as authors in [11] says, derived from qualitative considerations, thus if we want to derive a method to define an optimal threshold from them, they should be properly analysed.

The ultimate reason behind a proper definition of a risk threshold is to decide until which point we want to push our willingness to take those risks.

Implications, deriving from the adoption of the different methods to retrieve the risk, affect not only the different expressiveness capabilities of the risk policy but also the performances, in terms of time, that the system offers.

The performance analysis, regarding the time required by a policy evaluation process, which uses risk informations, retrieved by CAIRIS to take an authorization decision, has been performed in Chapter 7 and the results that have been obtained clearly show how the expressiveness of a policy characterizes its evaluation process. Indeed, when adopting API 1, which allows just the specification of the asset as a parameter, the operations that have to be performed in order to compute the highest risk score are much more, since no discrimination about threats and environments has been done, thus the evaluation times are higher. When adopting API 2, a further specification of the threat is allowed, thus permitting a finer grained decision, with a number of operations involved which mainly depends on the number of environments identified in the model and for which a threat has been identified. Similarly, API 3 allows the specification of a second parameter, beside the asset, which this time is the environment. This allows to retrieve the highest risk for a resource in a specific environment, helping the system to be aware of the most dangerous threats identified in a context, requiring a number of operations to perform which depends on the number of threats present in the database for that environment. Finally, the last supported risk retrieval method is represented by API 4, which allows to identify a risk by specifying the asset, the threat and the environment and representing in this way the finest grained way to identify a specific risk and also the fastest way to retrieve a risk since only indexing operations, which exploit the inner

structure of the database, are needed and no comparisons about risks have to be done in order to find the maximum.

Beside these variable time's components which change according to the kind of risk policy has been defined, it has been observed that the main factor affecting the whole policy evaluation time is given by the authentication part, which more or less takes one third of the time.

It is worth to say that the results, in terms of time, of the performances obtained, are also characterized by the core architecture that SAFAX adopts and over which the development of the *CAIRIS module*, to support the integration with the CAIRIS tool, took place, in such a way to obtain the desired risk-based authorization framework. The fact of implementing each XACML component separately as a standalone web service, in a distributed manner, limits in some way the performances of the system in respect to the situation in which all the functionalities of the components would be implemented in a centralized way, but, on the other end, allows to have a system easily customizable and with great extensibility and integrability properties.

## 8.2 Future works

An interesting direction that could be taken, in order to further enhance the developed system, could involve the definition of some alternative way of taking the final decision, based on the risks score retrieved. As already seen, the current decision method implements a threshold based solution, while some other ideas about possible alternatives have been suggested, in Chapter 2, when performing the state of the art analysis concerning risk-based access control systems.

A first interesting idea would be to implement a system able to balance both the concepts of risk and benefit (analysed in section 2.2). The idea of considering benefits together with risks and compare them in order to take an authorization decision, surely, bring us to a much more dynamic solution, able not only to consider the negative consequences of possible threats but also the positive implications deriving from an action. In this way, it would be clearer to understand when benefits outweigh risk in order to be more confident when taking a decision. In order to be possible to integrate an approach of this kind, in the system developed in this thesis work, we should probably consider to extend the usage of CAIRIS to support not only the management of risk parameters but of benefits related parameters as well. Since benefits, like risks, are strictly related to resources, it would be important, in a situation of this kind, to enhance the same system, which provides the risk scores, to provide the benefits scores too, thus avoiding to delegate this task to a third party, in such a way, for those values, to be compliant and homogeneous.

Actually, the functionalities that CAIRIS offers are not so distant from this kind of perspective, in fact, beside the development of risk models, it is able to support also the creation of task models as well, that cope with usability concepts. As it is now, tasks' related benefits are just descriptive and not quantified, but probably a similar quantification method used in the task analysis development, proposed in [28], could be adopted. There would be surely the need for developing a set of APIs, that given some predefined parameters, would be able to retrieve benefits scores. Furthermore, it would be important to define a way of associating benefits implications to the risks identified. According to the current way of managing risks, vulnerabilities are assigned to assets, at the same time threats are assigned to assets as well, finally a risk is computed by linking threats to vulnerabilities. Maybe, a possible association could be derived between benefits and vulnerabilities, since vulnerabilities may represent sensitive actions that could be performed over assets and from which both positive and negative consequences may arise. Although from a semantic point of view, the fact of associating benefits to vulnerabilities might sound strange, the approach could work within the CAIRIS environment, where there is not an explicit way of defining actions, that users perform over resources, but those actions might implicitly be taken into consideration when defining vulnerabilities. In order to clarify the concept, let's suppose to specify, as a vulnerability, *configuration file's modification*: it is a vulnerability possibly identifiable in a certain environment,

since a threat could arise from it, and, at the same time, it provides informations about the kind of access is related to the vulnerability involving a resource.

A second possible alternative to the threshold based mechanism, involves the adoption of a trust service, able to compute a trust score to integrate with the risk.

SAFAX already supports two mechanisms which allow to evaluate a policy considering the trust of the subjects: a Flow based Reputation Service [69] and an Evidence based Reputation Service [71]. The former service, has been implemented with the purpose of enhancing the traditional reputation models ([8], [41], [57]) by using a metric that gives absolute values to the users involved in a system, instead of just a ranking. The latter, is of particular interest because it creates a Flow based reputation model with uncertainty by putting together the advantages of both the two prominent reputation models present in literature, which are the Flow based model and the Subjective Logic [37].

The adoption of reputation systems allows to capture the actual trustworthiness of a target, where, clearly, the quality of the reputation values depends on the amount and the accuracy of informations used for its computation.

As asserted by [38], the main difference between trust and reputation systems stands in the fact that trust systems produce a score which reflects a subjective view of the relying party over the entity's trustworthiness, whereas reputation systems produce a score, assigned by the whole community, which represents the public reputation's view of the entity.

By using a reputation system within a specific environment we can make the two concepts converge and we can think about using the reputation score, as a value of trust that the system puts in the subject in that specific context, together with the risk score of a resource. In this way, we would be able to compute a more accurate metric when taking the final evaluation decision, in compliance with the policy.

It would be really interesting to find a way of exploiting, for example, the solution proposed in [66], according to which both risk and trust values should rely on a local reward history and a local penalty history, in such a way of creating a level of confidence which evolves in the time. This approach would be probably compliant with the two reputation services supported by SAFAX, that, in order to work properly, need informations captured by past transactions, indeed as asserted by [71]: “The basic idea underlying reputation is that a user's past experience as well as the experience of other users influences his decision whether to repeat this interaction in the future”. Clearly, how to consider, in CAIRIS, the history element is an issue that should be addressed.

The risk and trust values, then, could be compared or weighted according to some defined logic, in such a way to obtain the desired risk and trust authorization system.

By adopting a reputation service implemented by SAFAX, we would be able to cope with all the requirements identified by [Add. Req.4](#) in Chapter 3, which allow to properly implement a trust estimation module. Indeed, in order to support this kind of service, the SAFAX database supports the storage and the retrieval of trust informations, addressing in this way [Add. Req.4.1](#) and [Add. Req.4.2](#). The two reputation services have been implemented by two web services, installed on the SAFAX server, addressing [Add. Req.4.7](#), that in order to be exploited, can be contacted by means of specific APIs, in this way addressing [Add. Req.4.6](#). The logic which stands behind the implementation of the core part of the reputation mechanism, involving parameters and functions, and addressing [Add. Req.4.3](#) and [Add. Req.4.4](#), is explained in detail in [69] and [71]. In particular, through the SAFAX GUI, it is possible to upload the file containing a set of history transactions associated to scores that users have assigned to these transactions. Moreover there is the possibility of defining the value of alpha for computing reputation, which represents a weight parameter for the importance of indirect versus direct evidence, the initial vector representing the initial reputation values for the subjects involved and finally the function to adopt when representing the belief a subject has when looking at the trust score of another subject. In this way we can address [Add. Req.4.5](#).

In order to successfully cope with all the identified requirements supporting a risk and trust based

authorization service, there would be the need of implementing a risk & trust based decision module as addressed by [Add. Req.5](#). This module should implement the necessary decision logic for making coexist both risk and trust scores retrieved. This logic could be either implemented in a new customized component or it could be integrated within one of the two already existing services, which are the *CAIRIS module* and the *Reputation Trust Service* and letting the two service communicate in order to exchange risk and trust informations, addressing in this way [Add. Req.5.1](#). By adopting this second solution we would automatically address also [Add. Req.5.3](#) and [Add. Req.5.4](#). In order to complete the implementation, a decision function that creates a relation between risk and trust scores should be implemented, with the purpose of returning the final risk authorization decision to the PDP and coping with [Add. Req.5.2](#). For example, we could think about either a simple or a weighted comparison between the two values.

An other possibility of enhancement, for the risk-based authorization system proposed, is identified, by the introduction of an obligation service, able to mitigate the risk that the system is willing to take. To this purpose we could think about upgrading the threshold into a region of risk. In this way, we can fix a lower bound for the minimum amount of risk that could bring some damage in our system and an upper bound for the maximum amount of risk we are willing to accept.

When developing solutions of this kind in an organization, we need to keep in mind that the ultimate goal is to achieve results that wouldn't be possible with a traditional role-based mechanism, thus allowing, in some cases, the presence of some percentage of risk when authorizing an access, and helping, in this way, a user to accomplish his tasks. On the other end, in order to maintain the inner security of the system we need some way of controlling risks, meaning that we should allow the access to a resource in presence of some amount of risk, only at certain conditions. Those conditions can be identified as mitigation measures, which consist in some form of obligations a user should fulfil, in order to mitigate the risk associated to the action he wants to perform.

Between the two bounds, several levels of risk could be identified and to each one of them a specific obligation should correspond; the higher the risk identified by a level and the more relevant an obligation should be, in order to increase the responsibility feeling in a user and his risk perception, and, at the same time, keeping the focus on the organisation's optimal goal, which is to give the possibility to users to successfully accomplish their tasks.

What an obligation should be in practice, clearly, strictly depends on the scenario in which the authorization system works. For example, an obligation could be a notification that a user has to submit to a responsible entity, which keeps track of all the transactions in order to verify, in a second moment, their correctness and effectiveness; alternatively it could represent a sort of price to pay according to a market based mechanism in order to purchase the permission to perform an action. The definition and the implementation of the supported obligations is addressed by [Add. Req.6.1](#).

SAFAX already implements an approach of this kind, applied to a UCON scenario, according to which the authorization control over the resources is monitored during the whole usage time of a resource, within a session. The implemented scenario, in particular, simulates the phone's credit mechanism, which allows a user to be authorized to make a call and sustain this call until his credit is positive. A credit is initially assigned to each user and during the whole duration of a call, the credit is partially decreased, thus a way for continuously monitor the user's authorization, in terms of sufficient credit, is needed.

A similar approach could be implemented by integrating it with the aforementioned proposal of introducing the trust concept inside the evaluation process: after a policy evaluation process has been completed and verified, the trustworthiness of the user could be either rewarded or penalized in such a way to create a trust score that could be used to "buy", in the future, the possibility of performing an action at the price of the obligation, thus working as a mitigating factor for the risk and addressing [Add. Req.6.2](#). In this case, obligations would represent the minimum amount of trust, requested by a system, in order to perform an action.

Since the PEP is the module which stands directly between the user and the authorization service

and is responsible of enforcing the final decision, taken by the system, it is generally considered as the most suitable component, in the XACML architecture, for implementing the obligation service. Alternatively, the obligation service could be implemented as a separate component that interacts, by means of ad hoc APIs with the PEP, helping it to enforce the final decision. The implementation specifications supporting such a system will cope with [Add. Req.6.3](#) and [Add. Req.6.4](#).

# Bibliography

- [1] *2017 Data Breach Investigations Report*. Verizon Enterprise. 74 pp. URL: [http://www.verizonenterprise.com/resources/reports/rp\\_DBIR\\_2017\\_Report\\_en\\_xg.pdf](http://www.verizonenterprise.com/resources/reports/rp_DBIR_2017_Report_en_xg.pdf).
- [2] *ACME Water Specification Exemplar*. Nov. 2017. URL: [https://cairis.org/ACME\\_Water/](https://cairis.org/ACME_Water/).
- [3] A. Adams and M. A. Sasse. «Users are not the enemy». In: *Communications of the ACM* 42.12 (1999), pp. 40–46. DOI: [10.1145/322796.322806](https://doi.org/10.1145/322796.322806).
- [4] Ali Ahmed and Ning Zhang. «A context-risk-aware access control model for ubiquitous environments». In: *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*. IEEE. 2008, pp. 775–782. DOI: [10.1109/IMCSIT.2008.4747331](https://doi.org/10.1109/IMCSIT.2008.4747331).
- [5] A. Almutairi et al. «A distributed access control architecture for cloud computing». In: *IEEE software* 29.2 (2012), pp. 36–44. DOI: [10.1109/MS.2011.153](https://doi.org/10.1109/MS.2011.153).
- [6] A. Armando et al. «Balancing trust and risk in access control». In: *OTM Confederated International Conferences 'On the Move to Meaningful Internet Systems'*. Springer. 2015, pp. 660–676. DOI: [10.1007/978-3-319-26148-5\\_45](https://doi.org/10.1007/978-3-319-26148-5_45).
- [7] A. Armando et al. «Risk-based privacy-aware information disclosure». In: *International Journal of Secure Software Engineering (IJSSE)* 6.2 (2015), pp. 70–89. DOI: [10.4018/IJSSE.2015040104](https://doi.org/10.4018/IJSSE.2015040104).
- [8] Sergey B. and Lawrence P. «The anatomy of a large-scale hypertextual Web search engine». In: *Computer Networks and ISDN Systems* 30.1 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: [10.1016/S0169-7552\(98\)00110-X](https://doi.org/10.1016/S0169-7552(98)00110-X).
- [9] *Barry persona*. Nov. 2017. URL: <https://cairis.org/barry>.
- [10] R. Buyya, R. Ranjan, and R. N. Calheiros. «Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services». In: *International Conference on Algorithms and Architectures for Parallel Processing*. Springer. 2010, pp. 13–31. URL: <https://arxiv.org/ftp/arxiv/papers/1003/1003.3920.pdf>.
- [11] Theodore C. and van der Gaag L. C. «Sensitivity Analysis for Threshold Decision Making with Dynamic Networks». In: *CoRR* (2012). arXiv: [1206.6818](https://arxiv.org/abs/1206.6818). URL: <http://arxiv.org/abs/1206.6818>.
- [12] *CAIRIS web site*. Nov. 2017. URL: <https://cairis.org/>.
- [13] J. W. Castro, S. T. Acuña, and N. Juristo. «Integrating the personas technique into the requirements analysis activity». In: *Computer Science, 2008. ENC'08. Mexican International Conference on*. IEEE. 2008, pp. 104–112. DOI: [10.1109/ENC.2008.40](https://doi.org/10.1109/ENC.2008.40).

- [14] D. W. Chadwick et al. «My private cloud overview: a trust, privacy and security infrastructure for the cloud». In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE. 2011, pp. 752–753. DOI: [10.1109/CLOUD.2011.113](https://doi.org/10.1109/CLOUD.2011.113).
- [15] A. Chen et al. «A Dynamic Risk-Based Access Control Model for Cloud Computing». In: *Big Data and Cloud Computing (BDCloud), Social Computing and Networking (Social-Com), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conferences on*. IEEE. 2016, pp. 579–584. DOI: [10.1109/BDCloud-SocialCom-SustainCom.2016.90](https://doi.org/10.1109/BDCloud-SocialCom-SustainCom.2016.90).
- [16] L. Chen and J. Crampton. «Risk-aware role-based access control». In: *International Workshop on Security and Trust Management*. Springer. 2011, pp. 140–156. URL: <http://www.isg.rhul.ac.uk/~jason/Pubs/stm11.pdf>.
- [17] L. Chen et al. «Obligations in risk-aware access control». In: *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*. IEEE. 2012, pp. 145–152. DOI: [10.1109/PST.2012.6297931](https://doi.org/10.1109/PST.2012.6297931).
- [18] P. Cheng et al. «Fuzzy multi-level security: An experiment on quantified risk-adaptive access control». In: *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE. 2007, pp. 222–230. DOI: [10.1109/SP.2007.21](https://doi.org/10.1109/SP.2007.21).
- [19] International Electrotechnical Commission et al. «Functional safety of electrical/electronic/programmable electronic safety related systems». In: *IEC 61508* (2000).
- [20] IEEE Standards Coordinating Committee et al. «IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos». In: *CA: IEEE Computer Society* 169 (1990). DOI: [10.1109/IEEESTD.1990.101064](https://doi.org/10.1109/IEEESTD.1990.101064).
- [21] *CORAS web site*. Nov. 2017. URL: <http://coras.sourceforge.net>.
- [22] Ferraiolo D. and Kuhn R. «Role-based access controls». In: *Proceedings of 15th NIST-NCSC National Computer Security Conference*. Vol. 563. Baltimore, Maryland: NIST-NCSC. 1992. URL: <https://csrc.nist.gov/CSRC/media/Publications/conference-paper/1992/10/13/role-based-access-controls/documents/ferraiolo-kuhn-92.pdf>.
- [23] Nguyen Ngoc Diep et al. «Contextual Risk-Based Access Control». In: *Security and Management 2007* (2007), pp. 406–412. URL: [https://www.researchgate.net/profile/Heejo\\_Lee/publication/221199840\\_Contextual\\_Risk-Based\\_Access\\_Control/links/5590a78f08ae15962d8c53a9.pdf](https://www.researchgate.net/profile/Heejo_Lee/publication/221199840_Contextual_Risk-Based_Access_Control/links/5590a78f08ae15962d8c53a9.pdf).
- [24] D. R. Dos Santos, C. M. Westphall, and C. B. Westphall. «A dynamic risk-based access control architecture for cloud computing». In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE. 2014, pp. 1–9. DOI: [10.1109/NOMS.2014.6838319](https://doi.org/10.1109/NOMS.2014.6838319).
- [25] A. I. Egner et al. «An Authorization Service for Collaborative Situation Awareness». In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. 2016, pp. 136–138. ISBN: 978-1-4503-3935-3. DOI: [10.1145/2857705.2857740](https://doi.org/10.1145/2857705.2857740).
- [26] S. Faily. «A framework for usable and secure system design». PhD thesis. University of Oxford, 2011. URL: <http://www.cs.ox.ac.uk/files/4045/thesis.pdf>.
- [27] S. Faily and I. Fléchaïs. «A meta-model for usable secure requirements engineering». In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*. ACM. 2010, pp. 29–35. DOI: [10.1145/1809100.1809105](https://doi.org/10.1145/1809100.1809105).
- [28] S. Faily and I. Fléchaïs. «Analysing and visualising security and usability in IRIS». In: *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*. IEEE. 2010, pp. 543–548. DOI: [10.1109/ARES.2010.28](https://doi.org/10.1109/ARES.2010.28).

- [29] S. Faily and I. Fléchaïs. «Software for interactive secure systems design: Lessons learned developing and applying cairis». In: *Proceedings of BCS HCI 2012 Workshops: Designing Interactive Secure Systems*. 2012. URL: <http://www.shamalfaily.com/wp-content/papercite-data/pdf/fafl121.pdf>.
- [30] S. Faily and I. Fléchaïs. «User-centered information security policy development in a post-stuxnet world». In: *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*. IEEE. 2011, pp. 716–721. DOI: [10.1109/ARES.2011.111](https://doi.org/10.1109/ARES.2011.111).
- [31] S. Faily et al. «Model-driven architectural risk analysis using architectural and contextualised attack patterns». In: *Proceedings of the Workshop on Model-Driven Security*. ACM, 2012, 3:1–3:6. DOI: [10.1145/2422498.2422501](https://doi.org/10.1145/2422498.2422501).
- [32] D. Fall et al. «Risk adaptive authorization mechanism (RAAdAM) for cloud computing». In: *Journal of Information Processing* 24.2 (2016), pp. 371–380. DOI: [10.2197/ipsjjip.24.371](https://doi.org/10.2197/ipsjjip.24.371).
- [33] N. Falliere, L. O. Murchu, and E. Chien. «W32. stuxnet dossier». In: *White paper, Symantec Corp., Security Response* 5.6 (2011). URL: [https://www.symantec.com/content/en/us/enterprise/media/security\\_response/whitepapers/w32\\_stuxnet\\_dossier.pdf](https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf).
- [34] IBM - Rational DOORS web page. Nov. 2017. URL: <http://www-03.ibm.com/software/products/it/ratidoor>.
- [35] ANSINCITS. «INCITS 359-2004. American National Standard for Information Technology- Role Based Access Control, American National Standards Institute». In: *Inc., NY, USA* (2004).
- [36] «ISO 9241-11. Ergonomic requirements for office work with visual display terminals (VDTs) –Part 11: Guidance on usability». In: *The international organization for standardization* 45 (1998).
- [37] A. Jøsang. «A logic for uncertain probabilities». In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 9.03 (2001), pp. 279–311. URL: <http://eprints.qut.edu.au/7204/1/Jos2001-IJUFKS.pdf>.
- [38] A. Jøsang, R. Ismail, and C. Boyd. «A survey of trust and reputation systems for online service provision». In: *Decision support systems* 43.2 (2007), pp. 618–644. DOI: [10.1016/j.dss.2005.05.019](https://doi.org/10.1016/j.dss.2005.05.019).
- [39] R. Kainda, I. Fléchaïs, and A. Roscoe. «Security and usability: Analysis and evaluation». In: *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*. IEEE. 2010, pp. 275–282. DOI: [10.1109/ARES.2010.77](https://doi.org/10.1109/ARES.2010.77).
- [40] S. P. Kaluvuri et al. «SAFAX—an extensible authorization service for cloud environments». In: *Frontiers in ICT* 2 (2015), p. 9. DOI: [10.3389/fict.2015.00009](https://doi.org/10.3389/fict.2015.00009).
- [41] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. «The eigentrust algorithm for reputation management in p2p networks». In: *Proceedings of the 12th international conference on World Wide Web*. ACM. 2003, pp. 640–651. DOI: [10.1145/775152.775242](https://doi.org/10.1145/775152.775242).
- [42] L. Krautsevich et al. «Risk-aware usage decision making in highly dynamic systems». In: *Internet Monitoring and Protection (ICIMP), 2010 Fifth International Conference on*. IEEE. 2010, pp. 29–34. DOI: [10.1109/ICIMP.2010.13](https://doi.org/10.1109/ICIMP.2010.13).
- [43] T. Kurze et al. «Cloud federation». In: *CLOUD COMPUTING 2011* (2011), pp. 32–38. URL: [http://www.aifb.kit.edu/images/0/02/Cloud\\_Federation.pdf](http://www.aifb.kit.edu/images/0/02/Cloud_Federation.pdf).

- [44] R. Laborde et al. «An extensible XACML authorization web service: Application to dynamic web sites access control». In: *Signal-Image Technology & Internet-Based Systems (SITIS), 2009 Fifth International Conference on*. IEEE. 2009, pp. 499–505. DOI: [10.1109/SITIS.2009.83](https://doi.org/10.1109/SITIS.2009.83).
- [45] H. Lakshmi et al. «Risk based access control in cloud computing». In: *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on*. IEEE. 2015, pp. 1502–1505. DOI: [10.1109/ICGCIoT.2015.7380704](https://doi.org/10.1109/ICGCIoT.2015.7380704).
- [46] A. X Liu et al. «Xengine: a fast and scalable XACML policy evaluation engine». In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 36. 1. ACM. 2008, pp. 265–276. DOI: [10.1145/1384529.1375488](https://doi.org/10.1145/1384529.1375488).
- [47] M. Lorch, D. Kafura, and S. Shah. «An XACML-based policy management and authorization service for globus resources». In: *Proceedings of the 4th International Workshop on Grid Computing*. IEEE Computer Society. 2003, p. 208. DOI: [10.1109/GRID.2003.1261718](https://doi.org/10.1109/GRID.2003.1261718).
- [48] R. Mahmudlu and N. den Hartog J. and Zannone. «Data Governance and Transparency for Collaborative Systems». In: *Data and Applications Security and Privacy XXX*. Springer International Publishing, 2016. DOI: [10.1007/978-3-319-41483-6\\_15](https://doi.org/10.1007/978-3-319-41483-6_15).
- [49] S. Marouf et al. «Adaptive reordering and clustering-based framework for efficient XACML policy evaluation». In: *IEEE Transactions on Services Computing* 4.4 (2011), pp. 300–313. DOI: [10.1109/TSC.2010.28](https://doi.org/10.1109/TSC.2010.28).
- [50] R McGraw. «Risk-adaptable access control (radac)». In: *Privilege (Access) Management Workshop. NIST–National Institute of Standards and Technology–Information Technology Laboratory*. 2009. URL: <https://csrc.nist.gov/csrc/media/events/privilege-management-workshop/documents/radac-paper0001.pdf>.
- [51] P. H. Meland et al. «SeaMonster: Providing tool support for security modeling». In: *Norsk informasjonssikkerhetskoneranse, NISK* (2008). URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.4582&rep=rep1&type=pdf>.
- [52] *MySQL Internals Manual*. Oracle. URL: <https://dev.mysql.com/doc/internals/en/files-in-innodb-sources.html>.
- [53] *NeuroGrid Exemplar*. Nov. 2017. URL: <https://cairis.org/NeuroGrid/>.
- [54] *NeuroGrid overview*. Nov. 2017. URL: <http://gtr.rcuk.ac.uk/projects?ref=G0300623>.
- [55] Q. Ni, E. Bertino, and J. Lobo. «Risk-based access control systems built on fuzzy inferences». In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM. 2010, pp. 250–260. DOI: [10.1145/1755688.1755719](https://doi.org/10.1145/1755688.1755719).
- [56] C. Pautasso and E. Wilde. «Why is the web loosely coupled?: a multi-faceted metric for service design». In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 911–920. DOI: [10.1145/1526709.1526832](https://doi.org/10.1145/1526709.1526832).
- [57] Lempel R. and Moran S. «The stochastic approach for link-structure analysis (SALSA) and the TKC effect». In: *Computer Networks* 33.1 (2000), pp. 387–401. ISSN: 1389-1286. DOI: [10.1016/S1389-1286\(00\)00034-7](https://doi.org/10.1016/S1389-1286(00)00034-7).
- [58] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Pearson Higher Education, 2004.
- [59] *SAFAX External Clients*. URL: [https://security1.win.tue.nl/safax/Documentation/safax\\_programmatically\\_integration-v0.2.pdf](https://security1.win.tue.nl/safax/Documentation/safax_programmatically_integration-v0.2.pdf).

- [60] *SAFAX Installation Guide*. URL: [http://security1.win.tue.nl/safax/Documentation/safax\\_installation\\_guide.pdf](http://security1.win.tue.nl/safax/Documentation/safax_installation_guide.pdf).
- [61] *SAFAX User Manual*. URL: [http://security1.win.tue.nl/safax/Documentation/safax\\_user\\_manual.pdf](http://security1.win.tue.nl/safax/Documentation/safax_user_manual.pdf).
- [62] P. Salini and S. Kanmani. «A novel method: Ontology-based security requirements engineering framework». In: *Emerging Trends in Engineering, Technology and Science (ICETETS), International Conference on*. IEEE. 2016, pp. 1–5. DOI: [10.1109/ICETETS.2016.7602982](https://doi.org/10.1109/ICETETS.2016.7602982).
- [63] A. Saltelli. «Sensitivity Analysis for Importance Assessment». In: *Risk Analysis* 22.3 (2002), pp. 579–590. ISSN: 1539-6924. DOI: [10.1111/0272-4332.00040](https://doi.org/10.1111/0272-4332.00040).
- [64] *SeaMonster download page*. Nov. 2017. URL: <http://sourceforge.net/projects/seamonster/>.
- [65] *Security Requirements Modeling Tool, Socio - Technical Security Modeling Language (rev 1.0)*. University of Trento. URL: [http://www.sts-tool.eu/download/documentation/Documentation\\_v.2.1.0/Manual\\_ModelingLanguage\\_v.2.1.0.pdf](http://www.sts-tool.eu/download/documentation/Documentation_v.2.1.0/Manual_ModelingLanguage_v.2.1.0.pdf).
- [66] R. A. Shaikh, K. Adi, and L. Logrippo. «Dynamic risk-based decision methods for access control systems». In: *computers & security* 31.4 (2012), pp. 447–464. DOI: [10.1016/j.cose.2012.02.006](https://doi.org/10.1016/j.cose.2012.02.006).
- [67] M. Sharma et al. «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-ICISS), 2012 IEEE 14th International Conference on*. IEEE. 2012, pp. 1047–1052. DOI: [10.1109/HPCC.2012.153](https://doi.org/10.1109/HPCC.2012.153).
- [68] R. W. Shirey. «Internet security glossary, version 2». In: (2007). URL: <https://tools.ietf.org/html/rfc4949>.
- [69] A. Simone, B. Skoric, and N. Zannone. «Flow-based reputation: more than just ranking». In: *International Journal of Information Technology & Decision Making* 11.03 (2012), pp. 551–578. URL: <https://security1.win.tue.nl/~zannone/publication/simo-skor-zann-12-IJITDM.pdf>.
- [70] G. Sindre and A. L. Opdahl. «Eliciting Security Requirements by Misuse Case». In: *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Pacific 2000), IEEE CS Press*. 2000. DOI: [10.1109/TOOLS.2000.891363](https://doi.org/10.1109/TOOLS.2000.891363).
- [71] B. Skoric, S. J. A. de Hoogh, and N. Zannone. «Flow-based reputation with uncertainty: evidence-based subjective logic». In: *International Journal of Information Security* 15.4 (Aug. 2016), pp. 381–402. ISSN: 1615-5270. DOI: [10.1007/s10207-015-0298-5](https://doi.org/10.1007/s10207-015-0298-5).
- [72] OASIS Standard. *eXtensible Access Control Markup Language (XACML) Version 3.0*. Jan. 2013. URL: <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>.
- [73] *STS-Tool web site*. Nov. 2017. URL: <http://www.sts-tool.eu>.
- [74] A. K. Thurimella and D. Janzen. «Metadoc Feature Modeler: A Plug-in for IBM Rational DOORS». In: *2011 15th International Software Product Line Conference*. IEEE. 2011, pp. 313–322. DOI: [10.1109/SPLC.2011.17](https://doi.org/10.1109/SPLC.2011.17).
- [75] D. Trivellato, N. Zannone, and S. Etalle. «GEM: A distributed goal evaluation algorithm for trust management». In: *Theory and Practice of Logic Programming* 14.3 (2014), 293?337. DOI: [10.1017/S1471068412000397](https://doi.org/10.1017/S1471068412000397).

- [76] Q. Wang and H. Jin. «Quantified risk-adaptive access control for patient privacy protection in health information systems». In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM. 2011, pp. 406–410. doi: [10 . 1145 / 1966913 . 1966969](https://doi.org/10.1145/1966913.1966969).
- [77] K. Yee. «Aligning security and usability». In: *IEEE Security & Privacy 2.5* (2004), pp. 48–55. doi: [10 . 1109 / MSP . 2004 . 64](https://doi.org/10.1109/MSP.2004.64).
- [78] L. Zhang, A. Brodsky, and S. Jajodia. «Toward information sharing: Benefit and risk access control (barac)». In: *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*. IEEE. 2006, 9–pp. doi: [10 . 1109 / POLICY . 2006 . 36](https://doi.org/10.1109/POLICY.2006.36).

# Appendix A

## Developer Manual

The purpose of this manual is to describe the code developed to implement the integration between CAIRIS and SAFAX tools. The project is going to be described package by package and file by file.

### A.1 “cairis”module

Folder named “cairis” represents the *CAIRIS module* implemented “ex novo” on SAFAX. This is a web service developed in JAVA and packaged as a WAR file; it represents the core part of the final integration. In the subfolder *src* it’s possible to find all the source files (.java) where the code for the implementation of the *CAIRIS module* is present, in particular they are organized in 4 packages:

1. /nl/tue/sec/cairis/db
2. /nl/tue/sec/cairis/engine
3. /nl/tue/sec/cairis/impl
4. /nl/tue/sec/cairis/util
5. /nl/tue/sec/cairis/ws

#### A.1.1 /nl/tue/sec/cairis/db

In this package 2 files are present:

**DBAbstraction.java:** In *DBAbstraction* class, methods supporting the connection to the MySQL SAFAX database (db) are present, together with the implementation of the upload, select, update and delete statements, which are used when performing the queries. The class has been declared *public* in order to be visible also outside its package, in such a way, for the implemented methods, to be invoked by other external classes. The methods defined inside *DBAbstraction* class are:

- **private static void connect:** It is a private static method, so it can be invoked just within this class and without having to create an instance of the class. It creates a *DBJerseyConfig* class’ instance which is used to retrieve all the configuration parameters needed to successfully perform the connection, such as: db host, db port, db name, db username, db password. These parameters are retrieved from the *config.db.properties* file.

- parameters: no input parameters.
- returns: no return value (void method).
- **private static boolean disconnect:** Performs the closure of the db connection.
  - parameters: no input parameters
  - returns: returns true if the closure of the db connection has been successfully performed, false otherwise.
- **public static int insertStatement:** It is a public method, so can be invoked also outside this class. By calling the *connect()* method, it retrieves the configuration parameters needed to open a connection towards the db and then it creates and executes the statement used to insert data inside the db. After that, the disconnect method is invoked and the connection is closed.
  - parameters:
    - \* String query: which is the query to be performed on the db.
    - \* List<String>variables: which is a list of all the parameters to insert inside the query, ordered as they appear in the query.
  - returns: returns an integer value greater than 0 if the statement has been executed correctly, 0 otherwise.
- **public static boolean updateStatement:** By calling the *connect()* method, it retrieves the configuration parameters needed to open a connection towards the db and then it creates and executes the statement used to upload data inside the db. After that, the disconnect method is invoked and the connection is closed.
  - parameters:
    - \* String query: which is the query to be performed on the db.
    - \* List<String>variables: which is a list of all the parameters to insert inside the query, ordered as they appear in the query.
  - returns: returns true if the statement has been executed correctly, false otherwise.
- **public static boolean deleteStatement:** By calling the *connect()* method, it retrieves the configuration parameters needed to open a connection towards the db and then it creates and executes the statement used to delete data inside the db. After that, the disconnect method is invoked and the connection is closed.
  - parameters:
    - \* String query: which is the query to be performed on the db.
    - \* List<String>variables: which is a list of all the parameters to insert inside the query, ordered as they appear in the query.
  - returns: returns true if the statement has been executed correctly, false otherwise.
- **public static ArrayList<String>selectRecords:** By calling the *connect()* method, it retrieves the configuration parameters needed to open a connection towards the db and then it creates and executes the statement used to select multiple tuples inside the db. After that, the disconnect method is invoked and the connection is closed.
  - parameters:
    - \* String query: which is the query to be performed on the db.

- \* List <String>variables: which is a list of all the parameters to insert inside the query, ordered as they appear in the query.
- returns: returns a list of String values which represent the tuples that have been selected from the db, *null* otherwise.
- **public static String selectRecord:** By calling the *connect()* method, it retrieves the configuration parameters needed to open a connection towards the db and then it creates and executes the statement used to select a single tuple inside the db. After that, the disconnect method is invoked and the connection is closed.
  - parameters:
    - \* String query: which is the query to be performed on the db.
    - \* List<String>variables: which is a list of all the parameters to insert inside the query, ordered as they appear in the query.
  - returns: returns a String value which represent the tuple that has been selected from the db, *null* otherwise.
- **public static ArrayList<String> selectColumns:** By calling the *connect()* method, it retrieves the configuration parameters needed to open a connection towards the db and then it creates and executes the statement used to extract, from each selected tuple, the attributes composing it. After that, the disconnect method is invoked and the connection is closed.
  - parameters:
    - \* String query: which is the query to be performed on the db.
    - \* List<String>variables: which is a list of all the parameters to insert inside the query, ordered as they appear in the query.
  - returns: returns a list of String value which represent the attributes of the tuples that have been selected from the db, *null* otherwise.

**DBFns.java** : In *DBFns* class, methods that perform queries to the SAFAX database are defined. The class has been defined *public* in order to be visible also outside its package and let the *public* methods defined in the class to be invoked. Furthermore, methods have been defined *static*, so that can be invoked without the need of creating an instance of the class. The methods defined inside *DBFns* class are:

- **public static int errorlog:** Insert the error in SAFAX *ext\_errorlog* table.
  - parameters:
    - \* String transactionID: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
    - \* String logHead: which represents an explicative header explaining the type of error.
    - \* String logMsg: which is just an additional message that can be added in order to give more infos about the error.
    - \* int level: which is an integer value representing the error log level (4 = Invalid Errors, 3 = Network Errors, 2 = DB Errors, 1 = Parsing Errors, 0 = Every other Error).
    - \* String component: which represents the SAFAX component experiencing the error.

- returns: An integer value representing the result of the *insertStatement* method in *DBAbstraction* class. Which means an integer value greater than 0 if the execution of the query was successful, 0 otherwise.
- **public static String getDemoIDfromTransactionID:** Retrieves the *demoid* from the *sfx\_transaction* table.
  - parameters:
    - \* String transactionID: it's the SAFAX *transaction-id* associated to the transaction performed.
  - returns: An String value representing the *demoid* .
- **public static ArrayList<String> getCAIRIScredentials:** Retrieves, from the *sfx\_cairis\_credentials* table, the CAIRIS credentials associated to the specified *demoid*.
  - parameters:
    - \* String demoID: it's the identifier of a demo.
  - returns: A list of String values representing, respectively, the ccid (identifier of the tuple), the CAIRIS username and the CAIRIS password; *null* otherwise.
- **public static String getCAIRISDB:** Retrieves, from the *sfx\_cairis\_db* table, the CAIRIS database associated to the specified CAIRIS account.
  - parameters:
    - \* String ccid: it's the identifier of the tuple representing a CAIRIS account.
  - returns: A String value representing the CAIRIS database associated to the CAIRIS account identified by the *ccid*.

The followings are methods created with the purpose of being used by the *CAIRIS module* to perform some tests retrieving the risk values from the local *sfx\_cairis\_risks* db. They could be useful either in preliminary configuration phases, just to have a simple overview of the risk retrieval mechanism, or if the CAIRIS' APIs are not working (so, actually, in the normal configuration of SAFAX, these methods are never invoked).

- **public static String getHighestRiskValuefromResource:** Retrieves the highest risk value given a resource.
  - parameters:
    - \* String resource: which represents the asset which we want to retrieve the risk of.
  - returns: A string which is the result of the *selectRecord* method in *DBAbstraction* class.
- **public static String getHighestRiskValuefromResourceGivenThreat:** Retrieves the highest risk value given a resource and a threat.
  - parameters:
    - \* String resource: which represents the asset which we want to retrieve the risk of.
    - \* String threat: which represents the threat associated to the asset.

- returns: A string which is the result of the *selectRecord* method in *DBAbstraction* class.
- **public static String getHighestRiskValuefromResourceGivenEnvironment:** Retrieves the highest risk value given a resource and an environment.
  - parameters:
    - \* String resource: which represents the asset which we want to retrieve the risk of.
    - \* String environment: which represents the environment within which we want to retrieve the risk.
  - returns: A string which is the result of the *selectRecord* method in *DBAbstraction* class.
- **public static String getHighestRiskValuefromResourceGivenThreatandEnvironment:** Retrieves the highest risk value given a resource, a threat and an environment.
  - parameters:
    - \* String resource: which represents the asset which we want to retrieve the risk of.
    - \* String threat: which represents the threat associated to the asset.
    - \* String environment: which represents the environment within which we want to retrieve the risk.
  - returns: A string which is the result of the *selectRecord* method in *DBAbstraction* class.

### A.1.2 /nl/tue/sec/cairis/engine

In this package, just 1 file is present:

**CairisEngine.java** : The *CairisEngine* class is a public class, where the method, in which resides the core part of the implementation engine of the *CAIRIS module*, is implemented. The class allows the adoption of two authentication methods implemented. Since, as default, the Basic HTTP authentication method is enabled, in order to use the simple authentication method, you should just uncomment the line which follows the comment string “SIMPLE SESSION\_ID AUTHENTICATION” and comment the part of the code between the comment strings “HTTP BASIC AUTHENTICATION” and “End HTTP BASIC AUTHENTICATION”. The method defined inside *CairisEngine* class is:

- **public static int coreExecute:** It’s the core logic that resides behind the functioning of the *CAIRIS module*. It invokes methods to retrieve CAIRIS credentials and database, to perform authentication on CAIRIS and to retrieve the risk.
  - parameters:
    - \* String CairisURL: Risk API URL.
    - \* String transID: it’s the SAFAX *transaction-id* representing the transaction performed.
    - \* String authURL: CAIRIS API to contact in order to obtain the *session-id*.
    - \* String CAIRISdb: the name of the CAIRIS db to contact.
  - returns: A positive integer value ( $\geq 0$ ) representing the value of risk, retrieved by CAIRIS, -1 otherwise.

### A.1.3 /nl/tue/sec/cairis/impl

In this package 3 files are present:

**AuthenticationMethods.java** : The *AuthenticationMethods* class is a public class, where methods that support the two kinds of authentication supported by CAIRIS are implemented. The methods defined inside *AuthenticationMethods* class are:

- **public static ClientResponse httpBasicAuth**: Allows to perform http basic authentication, by contacting the proper CAIRIS api, passed as parameter, through POST method, and giving it username and password in order to obtain a valid *session-id*.
  - parameters:
    - \* String username: username used to authenticate on CAIRIS.
    - \* String passwd: password used to authenticate on CAIRIS.
    - \* String authURL: CAIRIS API to contact in order to obtain the *session-id*.
  - returns: An object of type *ClientResponse* which represents the response to the invocation of the *authURL*.
- **public static ClientResponse simpleAuth**: Allows to directly retrieve the risk value from the CAIRIS API, passed as parameter by means of a standard and predefined authentication (*session\_id=test*) supported by CAIRIS.
  - parameters:
    - \* String cairisURL: the complete URL to invoke in order to retrieve the risk from CAIRIS.
  - returns: An object of type *ClientResponse* which represents the response to the invocation of the *cairisURL*.

**CAIRISRetrievalMethods.java**: The *CAIRISRetrievalMethods* class is a public class implementing the risk's retrieval method, used when HTTP Basic authentication is previously used and a valid *session-id* has been correctly retrieved. The method defined inside *CAIRISRetrievalMethods* class is:

- **public static ClientResponse getRisk**: Allows to retrieve the risk value from the CAIRIS API passed as parameter after that a valid *session-id* have been negotiated. The *session-id* is extracted from the response, received by CAIRIS, parsed and attached to the *CairisURL*.
  - parameters:
    - \* ClientResponse resp: it's the response, received from CAIRIS, to the HTTP basic authentication request.
    - \* String CairisURL: the complete URL to invoke in order to retrieve the risk from CAIRIS.
    - \* String CAIRISdbURL: the CAIRIS database URL API to contact in order to open the desired db.
    - \* String dbname: is the name of the CAIRIS database to open.
    - \* String transID: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.

- returns: An object of type *ClientResponse* which represents the response to the invocation of the *CairisURL*.
- **private static ClientResponse openCAIRISdb:** Allows to open the CAIRIS database passed as parameter. The method has been implemented as *private*, in such a way to be visible only within the class.
  - parameters:
    - \* String sessionCairis: the *session-id* retrieved from CAIRIS and to include in the database open API.
    - \* String CAIRISdbURL: the CAIRIS database URL API to contact in order to open the desired db.
    - \* String dbname: is the name of the CAIRIS database to open.
    - \* String transID: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
  - returns: An object of type *ClientResponse* which represents the response to the invocation of the *CAIRISdbURL*.

**EvaluationMethods.java** : The *EvaluationMethods* class is a public class implementing the threshold based risk's evaluation method. If, in the future, a new evaluation method will be implemented, it could be defined in this class. The method defined inside *EvaluationMethods* class is:

- **public static boolean evaluateThreshold:** Allows to compare the risk, passed as parameter, against a threshold, passed as parameter too.
  - parameters:
    - \* int risk: it's the risk that have been retrieved from CAIRIS.
    - \* int threshold: it's the threshold against which we compare the risk.
    - \* transID: it's the SAFAX transaction-id associated to the transaction performed, that allows to call methods for writing logs.
  - returns: True if the risk is below the threshold, false otherwise.

#### A.1.4 /nl/tue/sec/cairis/util

In this package 4 files are present:

**CairisUtil.java:** The *CairisUtil* class is a public class where utility functions can be implemented. First of all, the COMPKEY parameter is defined. In particular it is defined as *private*, since there is no reason why it should be accessible from outside this class, *final* because it is a value that will never change and *static* so that can be directly called without creating an instance of the class. The method defined inside *CairisUtil* class is:

- **public static String writeLog:** This method is used to contact the PAP component which is, among other things, responsible of storing and showing the logs relative to the evaluation mechanism of a XACML request, showing them in the Account Activity section of SAFAX application. The data are passed in the URL by means of a *Form* instance, the parameters passed in the *Form* object are: *clientcode* (a unique string that identifies the component which is writing the logs), *transactionid* (the SAFAX transaction-id), *level* (the log level), *message* (the message to write in the logs), *serviceid* (the denomination of the component: *nl:tue:sec:cairis*).

- parameters:
  - \* String transactionid: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
  - \* int level: it's the log level (3 = Network Messaging, 2 = Internal messages to the component, 1 = Interface level, 0 = Every other, -1 = Info, -2 = Warning, -3 = Error, -4 = Super Critical Errors).
  - \* String message: it's the message to write in the log.
- returns: A string representing the entity response if the PAP URL has been correctly contacted (code status = 200), an empty string otherwise.

**DataUtil.java:** The *DataUtil* class is a public class where JSON parsing methods are defined. The methods defined inside *DataUtil* class are:

- **public static JSONObject MapToJSON:** This method converts a *LinkedHashMap* into a *JSONObject*.
  - parameters:
    - \* *LinkedHashMap*<String, String> lmap: the map to be converted.
  - returns: a *JSONObject* object.
- **public static JSONArray MapToJSON:** This method converts an array of *LinkedHashMap* into a JSON Object.
  - parameters:
    - \* *ArrayList*<*LinkedHashMap*<String, String>> lmap: the array of maps to be converted.
  - returns: a *JSONArray* array of *JSONObject* objects.
- **public static ArrayList<LinkedHashMap<String,String>> JSONArrayToMap:** This method converts an array of JSON objects into an array of maps.
  - parameters:
    - \* *JSONArray* jArray: the JSON array to be converted.
  - returns: an *ArrayList* of *LinkedHashMap* maps.
- **public static LinkedHashMap<String,String> JSONToMap:** This method converts a JSON object into a map.
  - parameters:
    - \* *JSONObject* json: the JSON object to be converted.
  - returns: a *LinkedHashMap* map.
- **public static JSONArray StringToJSONArray:** This method converts a string into a JSON array.
  - parameters:
    - \* String jsonString: the string to be converted.
  - returns: a *JSONArray* array.
- **public static ArrayList<String> convertToList:** This method converts a sequence of strings into an array of strings.

- parameters:
  - \* String...args: the sequence of strings to be converted.
- returns: an *ArrayList* array of String objects.
- **public static JSONObject getJSONFromInt:** This method converts an integer into a JSON object.
  - parameters:
    - \* int response: the integer value to be converted.
  - returns: a *JSONObject* object.
- 
- **public static JSONObject getJSONFromDouble:** This method converts a double into a JSON object.
  - parameters:
    - \* double response: the double value to be converted.
  - returns: a *JSONObject* object.
- **public static JSONObject getJSONFromString:** This method converts a string into a JSON object.
  - parameters:
    - \* String response: the string to be converted.
  - returns: a *JSONObject* object.
- **public static JSONObject getJSONFromBool:** This method converts a boolean value into a *JSONObject*.
  - parameters:
    - \* Boolean response: the boolean value to be converted.
  - returns: a *JSONObject* object.
- **public static Response buildResponse:** This method builds a response to an invocation of an API, with a *JSONString* object.
  - parameters:
    - \* JSONObject json: the *JSONObject* to pass in the response.
  - returns: a Response object.
- **public static Response buildResponse:** This method builds a response to an invocation of an API, with a *JSONString* object.
  - parameters:
    - \* JSONArray json: the JSON array to pass in the response.
  - returns: a Response object.

**DBJerseyConfig.java:** The *DBJerseyConfig* class is a public class used to retrieve the configuration parameters in order to successfully connect to SAFAX MySQL db. The methods defined inside *DBJerseyConfig* class are:

- **public String getDBHost:** Allows to get the db host.
  - parameters: no input parameters.
  - returns: a string indicating the db host.
- **public String getDBPort:** Allows to get the db port.
  - parameters: no input parameters.
  - returns: a string indicating the db port.
- **public String getDB:** Allows to get the db name.
  - parameters: no input parameters.
  - returns: a string indicating the db name.
- **public String getDBUser:** Allows to get the db username.
  - parameters: no input parameters.
  - returns: a string indicating the db username.
- **public String getDBPassword:** Allows to get the db password.
  - parameters: no input parameters.
  - returns: a string indicating the db password.
- **public String getPropValues:** Allows to read and get the configuration parameters from the file where they are specified (*config.db.properties* file).
  - parameters: no input parameters.
  - returns: a string value that list all the parameters read from the file.

**LogUtil.java:** The *LogUtil* class is a public class which implements log methods. The methods defined inside *LogUtil* class are:

- **public static void errorlog:** it just invokes *errorlog* method in *DBFns* class.
  - parameters:
    - \* String transactionid: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
    - \* String header: represents an explicative header explaining the type of error.
    - \* String message: just an additional message that can be added in order to give more infos about the error.
    - \* int level: an integer value representing the error log level (4 = Invalid Errors, 3 = Network Errors, 2 = DB Errors, 1 = Parsing Errors, 0 = Every other Error).
  - returns: no return value (void method).
- **public static void writeLog:** it just invokes *writeLog* method in *CairisUtil* class.
  - parameters:
    - \* String transactionid: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
    - \* String message: just an additional message that can be added in order to give more infos about the error.
    - \* int level: an integer value representing the error log level (4 = Invalid Errors, 3 = Network Errors, 2 = DB Errors, 1 = Parsing Errors, 0 = Every other Error).
  - returns: no return value (void method).

### A.1.5 /nl/tue/sec/cairis/ws

In this package 1 file is present:

**CairisService.java:** The *CairisService* class is a public class where CAIRIS UDFs are specified. The core part of the implementation of these UDFs has been defined in the *CairisEngine* file, within *nl.tue.sec.cairis.engine* package. Indeed, each one of the UDFs, after a preliminary self-configuration step, invokes the *coreExecute* method, which follows a standard procedure to retrieve the final risk value from CAIRIS and thus can exploit a unified code. In this class, 3 *private final static* parameters are defined: *CAIRISHOME*, which represents the URL of the CAIRIS demo, *authURL*, which represents the URL of the API to contact in order to obtain a session in CAIRIS, and *CAIRISdb*, which represents the URL to contact in order to manage the relation with a CAIRIS database. This parameters are defined *private* because they are going to be used only within this class and *final*, since they will never change their value. The UDFs implemented in *CairisService* class are:

- **public Response asset**

- urn:bu:udf:cairis:risk:level:asset : It's the UDF that should be able to retrieve the highest risk for an asset.
- Path: api/risk\_level/asset/{asset\_name}/{threshold}/{transID}
- Method: GET
- Parameters:
  - \* asset\_name: it's the name of the resource which we want to retrieve the risk of.
  - \* threshold: it's the threshold against which evaluate the resource's risk.
  - \* transID: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
- Returns: a JSON Response encapsulating a boolean value, true if risk is below threshold, false otherwise.

- **public Response asset\_threat**

- urn:bu:udf:cairis:risk:level:asset:threat:type : It's the udf that should be able to retrieve the highest risk for an asset, given a threat type.
- Path: api/risk\_level/asset/threat\_type/{asset\_name}/{threat\_name}/{threshold}/{transID}
- Method: GET
- Parameters:
  - \* asset\_name: it's the name of the resource which we want to retrieve the risk of.
  - \* threat\_name: it's the name of the threat associated to the asset.
  - \* threshold: it's the threshold against which evaluate the resource's risk.
  - \* transID: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
- Returns: a JSON Response encapsulating a boolean value, true if risk is below threshold, false otherwise.

- **public Response asset\_environment**

- urn:bu:udf:cairis:risk:level:asset:environment : It's the udf that should be able to retrieve the highest risk for an asset, in a specific environment.
  - Path: `api/risk_level/asset/environment/{asset_name}/{environment}/{threshold}/{transID}`
  - Method: GET
  - Parameters:
    - \* `asset_name`: it's the name of the resource which we want to retrieve the risk of.
    - \* `environment`: it's the name of the environment in which we want to retrieve the resource's risk.
    - \* `threshold`: it's the threshold against which evaluate the resource's risk.
    - \* `transID`: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
  - Returns: a JSON Response encapsulating a boolean value, true if risk is below threshold, false otherwise.
- **public Response asset\_threat\_environment**
    - urn:bu:udf:cairis:risk:level:asset:threat:type:environment : It's the udf that should be able to retrieve the highest risk for an asset, in a specific environment, given a threat type.
    - Path: `api/risk_level/asset/threat_type/environment/{asset_name}/{threat_name}/{environment}/{threshold}/{transID}`
    - Method: GET
    - Parameters:
      - \* `asset_name`: it's the name of the resource which we want to retrieve the risk of.
      - \* `threat_name`: it's the name of the threat associated to the asset.
      - \* `environment`: it's the name of the environment in which we want to retrieve the resource's risk.
      - \* `threshold`: it's the threshold against which evaluate the resource's risk.
      - \* `transID`: it's the SAFAX *transaction-id* associated to the transaction performed, that allows to call methods for writing logs.
    - Returns: a JSON Response encapsulating a boolean value, true if risk is below threshold, false otherwise.

### A.1.6 The other “cairis”subfolders

In subfolder *resources* it's possible to find the *config.db.properties* file, which provides the configuration's informations needed to successfully interact with the SAFAX database. In subfolder *build* it's possible to find all the compiled files (.class) of the project. Finally, in *WebContent* subfolder it's possible to find the libraries included in the project and the *web.xml* file, which is the standard deployment descriptor for the Web application that the Web service is part of. It declares filters and servlets used by the service.

## A.2 “sfx”module

Folder named “sfx”represents the Web Service providing the graphical part of the project. Here HTML and Javascript files, needed to support the functioning of the application, are contained.

This Web Service was already present in the original project, so only the changes in the files that have been updated, to support the goal of this thesis work, will be reported here:

### A.2.1 WebContent/main.html

In this file, the HTML code has been updated in such a way to include, in the GUI, the *Risk* tab, where the user can insert and update the CAIRIS Settings. By searching for the word “CAIRIS”, inside the file, it is possible to see the updates that have been made. Mostly, they consist in a form where to insert username, password and database informations.

### A.2.2 WebContent/js/sfxmain.js

In this file, the javascript function called by the Save Settings button in *main.html* has been implemented. The function sends an Ajax request to the *sfxservice* module, contacting the *sfxservice/demo/edit/cairis/* API and passing as parameters the informations inserted in the form by the user, corresponding to the CAIRIS credentials and database informations. Then, it displays a message showing whether the result of the operations has been successful or not.

## A.3 “sfxservice”module

Folder named “sfxservice”represents the Web Service providing the interface between the web component of the project and the engine part, developed in java. This Web Service was already present in the original project, so only the changes in the files that have been updated, to support the goal of this thesis work, will be reported here:

### A.3.1 package nl.tue.sec.safax.sfxbe.db

**DBFns.java:** In *DBFns* class, methods that perform queries to the SAFAX database are defined. The class has been defined *public* in order to be visible also outside its package and let the *public* methods defined in the class to be invoked. Furthermore, methods have been defined *static*, so that can be invoked without the need of creating an instance of the class. The methods newly implemented are:

- **public static String setCairisCredentials:** The method sets into the *sfx\_cairis\_credentials* table the username and the password for the specific demo; if they already exist, it updates them.
  - parameters:
    - \* String cairisusername: the CAIRIS username.
    - \* String cairispwd: the CAIRIS password.
    - \* String demoid: the identifier of the demo.
  - returns: the identifier of the tuple for the set/updated CAIRIS account, *null* otherwise.
- **public static int setCairisDB:** The method sets into the *sfx\_cairis\_db* table the database to associate to a CAIRIS account specified in the *sfx\_cairis\_credentials* table. Whenever a database is added or updated, it becomes the default one.
  - parameters:

- \* String dbname: the CAIRIS database.
- \* String cairisid: the identifier of the tuple of the CAIRIS account to which associate the database.
- returns: an integer value representing the identifier of the tuple for the set/updated CAIRIS database, 0 if something in the process goes wrong.

### A.3.2 package nl.tue.sec.safax.sfxbe.impl

**DemoHandlerImpl.java:** The *DemoHandlerImpl* class is a public class where the implementation engine of methods in *DemoHandler.java* file is implemented. The method newly implemented is:

- **public String editCairis:** The method allows to invoke the database functions supporting the setting or the update of the CAIRIS credentials and database informations.
  - parameters:
    - \* String demoid: the demo identifier.
    - \* String uname: the SAFAX username.
    - \* String cairisusr: the CAIRIS username.
    - \* String cairispwd: the CAIRIS password.
    - \* String cairisdb: the CAIRIS database.
    - \* String projectid: the SAFAX project identifier.
    - \* String sessionid: the SAFAX user session identifier.
  - returns: a string value representing the result of the operation.

### A.3.3 package nl.tue.sec.safax.sfxbe.ws

**DemoHandler.java:** The *DemoHandler* class is a public class where the SAFAX API called by the javascript functions through Ajax request, are defined. The API newly implemented is:

- **public Response editCairis**
  - Path: /edit/cairis/{demoid}/{projectid}
  - Method: POST
  - URL parameters:
    - \* String demoid: the demo identifier.
    - \* String projectid: the SAFAX project identifier.
  - Path parameters:
    - \* String cairisusr: the CAIRIS username.
    - \* String cairispwd: the CAIRIS password.
    - \* String cairisdb: the CAIRIS database.
  - Cookie parameters:
    - \* String sessionid: the SAFAX user session identifier.
  - returns: a JSON Response encapsulating a string value representing the result of the operation, carried on by the *editCairis* method in *DemoHandlerImpl* class.

## A.4 db\_risk\_tables folder

In the folder “db\_risk\_tables” are present the scripts to create the SAFAX database tables needed in the project.

- **safax\_sfx\_cairis\_risks.sql**

The script creates a SAFAX risk table, named `sfx_cairis_risks`, that simulates the risks retrieved by CAIRIS and that can be used to test the *CAIRIS module* if, for example, the CAIRIS service is not available. In order to achieve this, a table with these columns is needed:

- resourceid (PrimaryKey, not null, Datatype: INT)
- resourcename (not null, Datatype: VARCHAR)
- threatname (Default: null, Datatype: VARCHAR)
- environmentname (Default: null, Datatype: VARCHAR)
- riskvalue (not null, Datatype: INT)

In order to successfully perform the test, you should uncomment the line of codes in *CairisService* class which are between the comment strings “TEST” and “End TEST”. Then, you should comment the part of code where the *coreExecute* method is invoked.

- **safax\_sfx\_cairis\_credentials.sql**

The script creates a table, named `sfx_cairis_credentials`, that contains the CAIRIS credentials associated to each demo in the application. In order to achieve this, a table with these columns is needed:

- ccid (PrimaryKey, not null, unsigned, zero fill, auto increment, Datatype: INT)
- cairisuname (not null, Datatype: VARCHAR)
- cairispwd (not null, Datatype: VARCHAR)
- demoid (not null, unsigned, zero fill, Datatype: INT)

- **safax\_sfx\_cairis\_db.sql**

The script creates a table, named `sfx_cairis_db`, that contains the CAIRIS databases (active and not active) associated with each credential. In order to achieve this, a table with these columns is needed:

- cdid (PrimaryKey, not null, unsigned, zero fill, auto increment, Datatype: INT)
- dbname (not null, Datatype: VARCHAR)
- ccid (not null, unsigned, zero fill, Datatype: INT)
- isactive (Default: 1, Datatype: TINYINT(1))

## A.5 example\_policies\_requests folder

In the folder “example\_policies\_requests” are present the XACML risk policies and requests that can be used to benefit from this Risk-based authorization mechanism, each one tests a different CAIRIS API:

- **risk\_policy\_asset.xml**: This is an example of Risk policy, which asks for a *resource-id* and an *action-id*. The two attributes' values are compared to the attributes with same *attribute-id* in the request according to *urn:oasis:names:tc:xacml:1.0:function:string-equal* function, which is just a string comparator function. In this way, the applicability of the policy is verified. In the condition of the rule, instead, *urn:bu:udf:cairis:risk:level:asset* UDF is called. The function takes as parameters the attributes specified inside, which in this case are the *resource-id* attribute specified in the request and an integer value representing the threshold. If the result of the condition in the first rule will be TRUE, then the effect will be PERMIT, if FALSE, then the second rule's effect will be DENY. (See figure A.1).
- **risk\_request\_asset.xml**: This is an example of Risk request, which wants to perform a certain action over a specified resource, so *resource-id* and *action-id* have to be specified. (See figure A.2).
- **risk\_policy\_asset\_env.xml**: This is an example of Risk policy, which asks for an *resource-id*, an *environment-id* and an *action-id*. The three attributes' values are compared to the attributes with same *attribute-id* in the request according to *urn:oasis:names:tc:xacml:1.0:function:string-equal* function, which is just a string comparator function. In this way, the applicability of the policy is verified. In the condition of the rule, instead, *urn:bu:udf:cairis:risk:level:asset:environment* UDF is called. The function takes as parameters the attributes specified inside, which in this case are the *resource-id* attribute and the *environment-id* attribute specified in the request and an integer value representing the threshold. If the result of the condition in the first rule will be TRUE, then the effect will be PERMIT, if FALSE, then the second rule's effect will be DENY. (See figure A.3).
- **risk\_request\_asset\_env.xml**: This is an example of Risk request, which wants to perform a certain action over a specified resource, in a specific context, so *resource-id*, *environment-id* and *action-id* have to be specified. (See figure A.4).
- **risk\_policy\_asset\_threat.xml**: This is an example of Risk policy, which asks for an *resource-id*, a *threat-id* and an *action-id*. The three attributes' values are compared to the attributes with same *attribute-id* in the request according to *urn:oasis:names:tc:xacml:1.0:function:string-equal* function, which is just a string comparator function. In this way, the applicability of the policy is verified. In the condition of the rule, instead, *urn:bu:udf:cairis:risk:level:asset:threat:type* UDF is called. The function takes as parameters the attributes specified inside, which in this case are the *resource-id* attribute and the *threat-id* attribute specified in the request and an integer value representing the threshold. If the result of the condition in the first rule will be TRUE, then the effect will be PERMIT, if FALSE, then the second rule's effect will be DENY. (See figure A.5).
- **risk\_request\_asset\_threat.xml**: This is an example of Risk request, which wants to perform a certain action over a specified resource, given a certain threat, so *resource-id*, *threat-id* and *action-id* have to be specified. (See figure A.6).
- **risk\_policy\_asset\_threat\_env.xml**: This is an example of Risk policy, which asks for an *resource-id*, a *threat-id*, an *environment-id* and an *action-id*. The four attributes' values are compared to the attributes with same *attribute-id* in the request according to *urn:oasis:names:tc:xacml:1.0:function:string-equal* function, which is just a string comparator function. In this way, the applicability of the policy is verified. In the condition of

the rule, instead, *urn:bu:udf:cairis:risk:level:asset:threat:type:environment* UDF is called. The function takes as parameters the attributes specified inside, which in this case are the *resource-id* attribute, the *threat-id* attribute and the *environment-id* attribute specified in the request and an integer value representing the threshold. If the result of the condition in the first rule will be TRUE, then the effect will be PERMIT, if FALSE, then the second rule's effect will be DENY. (See figure [A.7](#)).

- **risk\_request\_asset\_threat\_env.xml**: This is an example of Risk request, which wants to perform a certain action over a specified resource, given a certain threat, in a specific context, so *resource-id*, *threat-id*, *environment-id* and *action-id* have to be specified. (See figure [A.8](#)).

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides"
5     PolicyId="urn:nl:tue:sec:example:sample:demo:cairis:asset"
6     xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
7     http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd">
8
9     <Target>
10        <Resources>
11            <Resource>
12                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
13                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
14                        ICT_Application
15                    </AttributeValue>
16                    <ResourceAttributeDesignator
17                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
18                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
19                </ResourceMatch>
20            </Resource>
21        </Resources>
22        <Actions>
23            <Action>
24                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
25                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
26                        read
27                    </AttributeValue>
28                    <ActionAttributeDesignator
29                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
30                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
31                </ActionMatch>
32            </Action>
33        </Actions>
34    </Target>
35
36    <Rule Effect="Permit" RuleId="urn:nl:tue:sec:example:sample:demo:cairis:rule:1">
37
38        <Condition>
39            <Apply FunctionId="urn:bu:udf:cairis:risk:level:asset">
40                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
41                    <ResourceAttributeDesignator
42                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
43                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
44                </Apply>
45                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">
46                    7
47                </AttributeValue>
48            </Apply>
49        </Condition>
50    </Rule>
51    <Rule Effect="Deny" RuleId="urn:nl:tue:sec:example:sample:demo:cairis:rule:2" />
52
53 </Policy>

```

Figure A.1. risk\_policy\_asset.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
5   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
6
7   <Resource>
8     <Attribute
9       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
10      DataType="http://www.w3.org/2001/XMLSchema#string">
11       <AttributeValue>
12         ICT_Application
13       </AttributeValue>
14     </Attribute>
15   </Resource>
16
17   <Action>
18     <Attribute
19       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
20       DataType="http://www.w3.org/2001/XMLSchema#string">
21       <AttributeValue>
22         read
23       </AttributeValue>
24     </Attribute>
25   </Action>
26
27   <Environment />
28 </Request>
```

Figure A.2. risk\_request\_asset.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides"
5     PolicyId="urn:nl:tue:sec:example:sample:demo:cairis:asset:environment"
6     xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
7     http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd">
8
9     <Target>
10        <Resources>
11            <Resource>
12                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
13                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
14                        ICT_Application
15                    </AttributeValue>
16                    <ResourceAttributeDesignator
17                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
18                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
19                </ResourceMatch>
20            </Resource>
21            <Resource>
22                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
23                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
24                        Day
25                    </AttributeValue>
26                    <ResourceAttributeDesignator
27                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:environment-id"
28                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
29                </ResourceMatch>
30            </Resource>
31        </Resources>
32        <Actions>
33            <Action>
34                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
35                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
36                        read
37                    </AttributeValue>
38                    <ActionAttributeDesignator
39                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
40                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
41                </ActionMatch>
42            </Action>
43        </Actions>
44    </Target>
45
46    <Rule Effect="Permit" RuleId="urn:nl:tue:sec:example:sample:demo:cairis:rule:1">
47
48        <Condition>
49            <Apply FunctionId="urn:bu:udf:cairis:risk:level:asset:environment">
50                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
51                    <ResourceAttributeDesignator
52                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
53                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
54                </Apply>
55                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
56                    <ResourceAttributeDesignator
57                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:environment-id"
58                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
59                </Apply>
60                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">
61                    10
62                </AttributeValue>
63            </Apply>
64        </Condition>
65    </Rule>
66    <Rule Effect="Deny" RuleId="urn:nl:tue:sec:example:sample:demo:cairis:rule:2" />
67
68 </Policy>

```

Figure A.3. risk\_policy\_asset\_env.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
5   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
6
7   <Resource>
8     <Attribute
9       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
10      DataType="http://www.w3.org/2001/XMLSchema#string">
11       <AttributeValue>
12         ICT_Application
13       </AttributeValue>
14     </Attribute>
15     <Attribute
16       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:environment-id"
17       DataType="http://www.w3.org/2001/XMLSchema#string">
18       <AttributeValue>
19         Day
20       </AttributeValue>
21     </Attribute>
22   </Resource>
23
24   <Action>
25     <Attribute
26       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
27       DataType="http://www.w3.org/2001/XMLSchema#string">
28       <AttributeValue>
29         read
30       </AttributeValue>
31     </Attribute>
32   </Action>
33
34   <Environment />
35 </Request>
```

Figure A.4. risk\_request\_asset\_env.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides"
5     PolicyId="urn:nlt:ue:sec:examplesample:demo:cairis:asset:threat"
6     xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-open.org/
7     xacml/access_control-xacml-2.0-policy-schema-os.xsd">
8
9     <Target>
10    <Resources>
11    <Resource>
12    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
13    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
14    ICT_Application
15    </AttributeValue>
16    <ResourceAttributeDesignator
17    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
18    DataType="http://www.w3.org/2001/XMLSchema#string"/>
19    </ResourceAttributeDesignator>
20    </ResourceMatch>
21    </Resource>
22    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
23    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
24    Enumeration
25    </AttributeValue>
26    <ResourceAttributeDesignator
27    AttributeId="urn:oasis:names:tc:xacml:2.0:resource:threat-id"
28    DataType="http://www.w3.org/2001/XMLSchema#string"/>
29    </ResourceAttributeDesignator>
30    </ResourceMatch>
31    </Resource>
32    </Resources>
33    <Actions>
34    <Action>
35    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
36    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
37    read
38    </AttributeValue>
39    <ActionAttributeDesignator
40    AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
41    DataType="http://www.w3.org/2001/XMLSchema#string"/>
42    </ActionAttributeDesignator>
43    </ActionMatch>
44    </Action>
45    </Actions>
46    </Target>
47
48    <Rule Effect="Permit" RuleId="urn:nlt:ue:sec:examplesample:demo:cairis:rule:1">
49    <Condition>
50    <Apply FunctionId="urn:bu:udf:cairis:risklevel:asset:threat:type">
51    <ResourceAttributeDesignator
52    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
53    DataType="http://www.w3.org/2001/XMLSchema#string"/>
54    </ResourceAttributeDesignator>
55    </Apply>
56    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
57    <ResourceAttributeDesignator
58    AttributeId="urn:oasis:names:tc:xacml:2.0:resource:threat-id"
59    DataType="http://www.w3.org/2001/XMLSchema#string"/>
60    </ResourceAttributeDesignator>
61    </Apply>
62    </Condition>
63    <Rule Effect="Deny" RuleId="urn:nlt:ue:sec:examplesample:demo:cairis:rule:2" />
64    </Rule>
65    </Policy>
66
67
68

```

Figure A.5. risk\_policy\_asset\_threat.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
5   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
6
7   <Resource>
8     <Attribute
9       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
10      DataType="http://www.w3.org/2001/XMLSchema#string">
11       <AttributeValue>
12         ICT_Application
13       </AttributeValue>
14     </Attribute>
15     <Attribute
16       AttributeId="urn:oasis:names:tc:xacml:2.0:resource:threat-id"
17       DataType="http://www.w3.org/2001/XMLSchema#string">
18       <AttributeValue>
19         Enumeration
20       </AttributeValue>
21     </Attribute>
22   </Resource>
23
24   <Action>
25     <Attribute
26       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
27       DataType="http://www.w3.org/2001/XMLSchema#string">
28       <AttributeValue>
29         read
30       </AttributeValue>
31     </Attribute>
32   </Action>
33
34   <Environment />
35 </Request>
```

Figure A.6. risk\_request\_asset\_threat.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides"
5     PolicyId="urn:nlt:u:sec:examples:sample:demo:cairis:asset:threat:environment"
6     xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-open.org/
7     xacml/access_control-xacml-2.0-policy-schema-os.xsd">
8
9     <Target>
10        <Resources>
11            <Resource>
12                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
13                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
14                        ICT_Application
15                    </AttributeValue>
16                    <ResourceAttributeDesignator
17                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
18                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
19                    </ResourceMatch>
20                </Resource>
21            <Resource>
22                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
23                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
24                        Enumeration
25                    </AttributeValue>
26                    <ResourceAttributeDesignator
27                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:threat-id"
28                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
29                    </ResourceMatch>
30                </Resource>
31            <Resource>
32                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
33                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
34                        Day
35                    </AttributeValue>
36                    <ResourceAttributeDesignator
37                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:environment-id"
38                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
39                    </ResourceMatch>
40                </Resource>
41            </Resources>
42        <Actions>
43            <Action>
44                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
45                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
46                        read
47                    </AttributeValue>
48                    <ActionAttributeDesignator
49                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
50                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
51                    </ActionMatch>
52                </Action>
53            </Actions>
54        </Target>
55
56        <Rule Effect="Permit" RuleId="urn:nlt:u:sec:examples:sample:demo:cairis:rule:1">
57
58            <Condition>
59                <Apply FunctionId="urn:bu:udf:cairis:risklevel:asset:threat:type:environment">
60                    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
61                        <ResourceAttributeDesignator
62                            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
63                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
64                    </Apply>
65                    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
66                        <ResourceAttributeDesignator
67                            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:threat-id"
68                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
69                    </Apply>
70                    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
71                        <ResourceAttributeDesignator
72                            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:environment-id"
73                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
74                    </Apply>
75                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">
76                        10
77                    </AttributeValue>
78                </Apply>
79            </Condition>
80        </Rule>
81        <Rule Effect="Deny" RuleId="urn:nlt:u:sec:examples:sample:demo:cairis:rule:2" />
82    </Policy>

```

Figure A.7. risk\_policy\_asset\_threat\_env.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
5     http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
6
7     <Resource>
8         <Attribute
9             AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
10            DataType="http://www.w3.org/2001/XMLSchema#string">
11             <AttributeValue>
12                 ICT_Application
13             </AttributeValue>
14         </Attribute>
15         <Attribute
16             AttributeId="urn:oasis:names:tc:xacml:1.0:resource:threat-id"
17            DataType="http://www.w3.org/2001/XMLSchema#string">
18             <AttributeValue>
19                 Enumeration
20             </AttributeValue>
21         </Attribute>
22         <Attribute
23             AttributeId="urn:oasis:names:tc:xacml:1.0:resource:environment-id"
24            DataType="http://www.w3.org/2001/XMLSchema#string">
25             <AttributeValue>
26                 Day
27             </AttributeValue>
28         </Attribute>
29     </Resource>
30
31     <Action>
32         <Attribute
33             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
34            DataType="http://www.w3.org/2001/XMLSchema#string">
35             <AttributeValue>
36                 read
37             </AttributeValue>
38         </Attribute>
39     </Action>
40
41     <Environment />
42 </Request>
```

Figure A.8. risk\_request\_asset\_threat\_env.xml.

# Appendix B

## User Manual

The purpose of this manual is to enhance the utility of the SAFAX Installation Guide [60] and the SAFAX User Manual [61], adding the necessary steps in order to install, better understand and make use of the new *CAIRIS module* implemented for this thesis work. This manual has been organized in two sections, the first one is more like an installation guide, which has been thought to help users who are going to install locally SAFAX from scratch, the second one is a user guide, which instead has the purpose of giving to the user the necessary knowledge to appropriately use the tool.

### B.1 Installation Guide

All the references to chapters, sections, pages that are made in this Installation Guide refers to the SAFAX Installation Guide [60]. Pay attention that not every single step described in [60] is repeated in this guide, but only the improvements, changes and of course the additional steps necessary in order to make use of SAFAX tool with the integration supporting the interaction with the CAIRIS tool.

#### B.1.1 Tomcat installation

In *Prerequisites > Tomcat > Windows section* (pag. 7).

In the following steps, it is explained how to download and install Tomcat. Tomcat is needed both for the development and execution of SAFAX, being used in this project as a servlet container for Jersey web services.

1. Create a custom directory (It is suggested to create the custom directory in the workspace of the user account on Windows system).
2. Download the 32-bit or 64-bit zip file from <https://tomcat.apache.org/download-70.cgi>.
3. Extract the zip file in the custom directory (make sure after the extraction of tomcat zip that no spaces are present in the path, in order to avoid problems).

### B.1.2 Safax repository download via SVN

In *Development Environment* > SVN section (pag. 13).

In this section it is explained how to download SAFAX repository and all its components by using SVN service.

A new folder named *cairis* will be now available, which contains the CAIRIS service. It is a component which allows to have a risk-based access control mechanism by contacting CAIRIS application as an external trust service and relying on it for risk parameters retrieval.

### B.1.3 MySQL

In *Development Environment* > MySQL section (pag. 19).

In the following steps it is described how to create with MySQL Workbench a new user account, which is the one that the module of SAFAX will use when contacting the database.

Use MySQL Workbench to create a new database and importing SAFAX database schema:

1. Click *Create a new schema in the connected server* button (B.1).

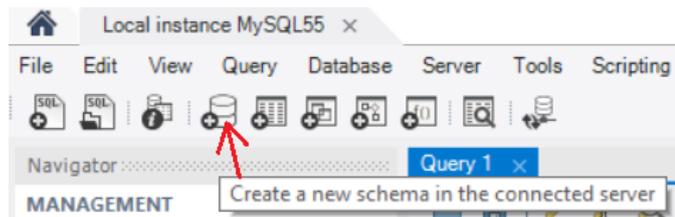


Figure B.1. Create a new schema.

2. Name the schema: safax.
3. Choose *latin1\_swedish\_ci* for the Collation (B.2).

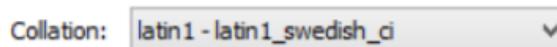


Figure B.2. Collation.

4. click *Open SQL Script* to import the SAFAX initial database. The script to use when importing tables can be found in the SAFAX SVN server.
5. Execute the script to create tables in the SAFAX database.
6. Create a user named vdJCh9F3 and assign the schema safax to this user, set a password for the user (the same password that the module are going to use when contacting the database) (B.3).
7. Assign to the user the necessary privileges needed to manage the tables (at least: SELECT, INSERT, UPDATE, DELETE, EXECUTE) (B.4).

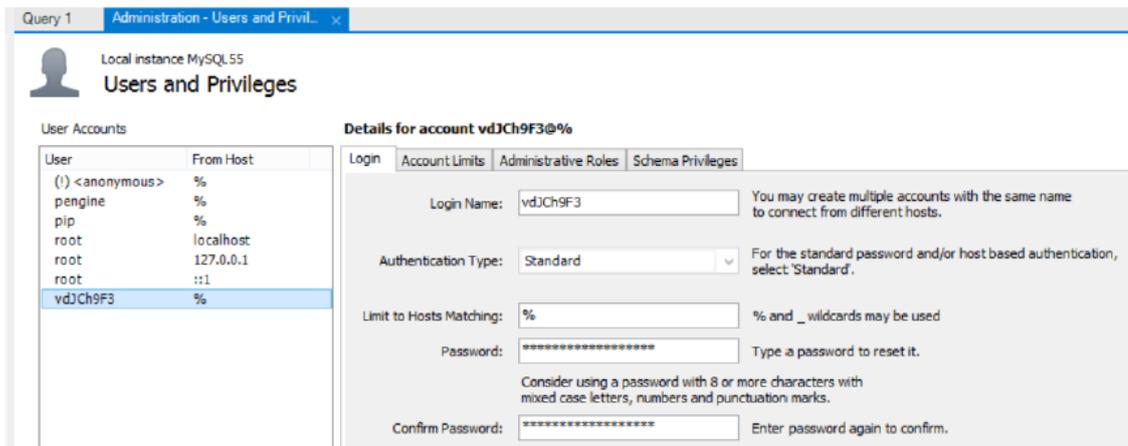


Figure B.3. User creation.

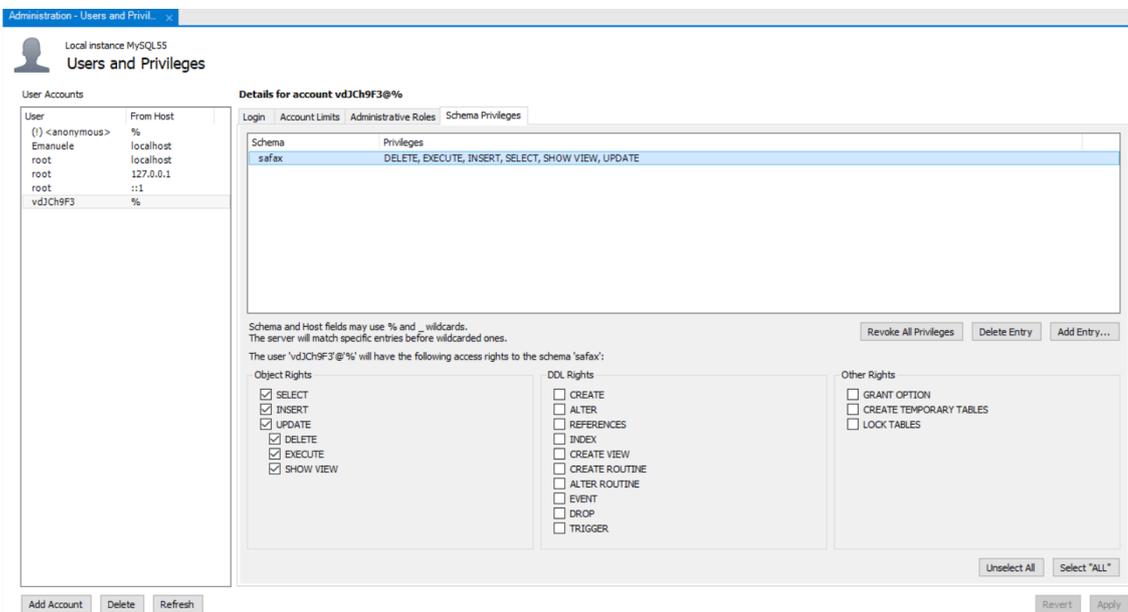


Figure B.4. User permission assignment.

## B.1.4 Execution Environment

In *Execution Environment* (pag. 20)

In the following steps it is described how to correctly setup the execution environment of SAFAX.

### Prerequisites

The following applications are needed to install the SAFAX development environment:

- JDK
- Tomcat
- MySQL

- MongoDB

See Section “Prerequisite” (pag. 5) for the installation of these applications.

## Service Deployment

Windows and Ubuntu

### 1. Some execution environment configuration:

- (a) (Only for Windows systems) Setup CATALINA\_HOME environmental variable.
  - Right-click on *My Computer* and select *Properties*.
  - Click on *Advanced system settings > Advanced > Environmental Variables*.
  - Under *System variables*, click on *New*.
  - In the Variable name enter: “CATALINA\_HOME”.
  - In the Variable value enter the path of the Tomcat’s installation folder.
  - Under *System Variables*, select variable *Path* and click on *Edit*.
  - In the Variable value append: “;%CATALINA\_HOME%\bin”.
- (b) (Only for Ubuntu systems) create a setenv.sh file in the bin folder of tomcat with the following content:

### 2. Update Tomcat users

Go to *tomcat-installed-folder/conf/tomcat-users.xml* and add the following lines:

```
<role rolename="tomcat"/>
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<role rolename="admin-gui"/>
<role rolename="admin-script"/>
<user username="tomcat"password="tomcat"roles="tomcat"/>
<user username="both"password="tomcat"roles="tomcat,role1"/>
<user username="role1"password="tomcat"roles="role1"/>
<user username="admin"password="admin"roles="manager-gui, manager-script, manager-jmx,manager-status, admin-gui, admin-script"/>
```

### 3. Change Tomcat port to 80

Go to *tomcat-installed-folder/conf/server.xml* and update the following lines:

```
<Connector port="80"protocol="HTTP/1.1"
connectionTimeout="20000"redirectPort="8443"/>
```

### 4. Deploy SAFAX web services to Tomcat server by using one of the two options:

- (a) Copying web application archive files (.war) into Tomcat webapps folder (*tomcat-installed-folder/webapps*). .war files can be found in SAFAX SVN server.
- (b) Using Tomcat’s manager application:
  - First start Tomcat  
Windows:
    - opening command prompt.

```

1  #!/bin/sh
2
3  # Java Heap Size = Place to store objects created by SAFAX web application.
4  # For a heavy service requested, insufficient Heap size will cause the popular
5  # java.lang.OutOfMemoryError
6
7  # Perm Gen Size = Place to store SAFAX web service loaded class definition and metadata
8  # If a large code-based service is loaded, the insufficient Perm Gen size will cause the popular
9  # Java.Lang.OutOfMemoryError: PermGen
10 # -XX:MaxPermSize - Set the maximum PermGen Size
11 # discourages address map swapping by setting Xms and Xmx to the same value?
12
13 export CATALINA_OPTS="$CATALINA_OPTS -server -Xms128m"
14 export CATALINA_OPTS="$CATALINA_OPTS -Xmx1024m"
15 export CATALINA_OPTS="$CATALINA_OPTS -XX:MaxPermSize=512m"
16
17 # Check for application specific parameters at startup
18
19 #If [ -r "$CATALINA_BASE/bin/appenv.sh" ]; then
20 #. "$CATALINA_BASE/bin/appenv.sh"
21 #fi
22
23 echo "Using CATALINA_OPTS:"
24 for arg in $CATALINA_OPTS
25 do
26 echo ">> " $arg
27 done
28
29 echo ""
30
31 export JAVA_OPTS="-Djava.library.path=/usr/local/lib"
32 echo "Using JAVA_OPTS:"
33 for arg in $JAVA_OPTS
34 do
35 echo ">> " $arg
36 done
37 done
38 echo "_____ "
39 echo ""

```

- typing `cd Tomcat-directory-name`.
- typing `.\bin\startup.bat` (or `%CATALINA_HOME%\bin\startup.bat`).
- typing `.\bin\shutdown.bat` (when later you will shutdown the server).
- typing `sudo .catalina.bat` (to debug while tomcat is running).

#### Ubuntu:

- opening a terminal.
- typing `cd /tomcat-installed-directory`.
- typing `sudo ./bin/startup.sh`.
- typing `sudo ./bin/shutdown.sh` (when later you will shutdown the server).
- typing `sudo ./catalina.sh run` (to debug while tomcat is running).

- Go to the Manager App (<http://localhost/manager/html>).

- iii. Login by using the username and password mentioned in tomcat-users.xml file (user: admin, passwd: admin).
  - Scroll down to the section “Deploy”.
  - Upload the .war files and click “Deploy”.

## Starting MongoDB application

### Windows

1. create the following folder: C:/data/db
2. open a terminal and type: cd C:/Program Files/MongoDB/Server/3.2/bin.
3. run mongoDB application by typing on the terminal: mongod.
4. by default, mongoDB server will start at port 27017.

### Linux

1. open a terminal.
2. start MongoDB service by typing: sudo service mongod start.
3. check if MySQL is running by checking the contents of the log file at: /var/log/mongodb/mongod.log for a line saying: [initandlisten] waiting for connections on port <port>.

## B.2 User Guide

All the references to chapters, sections, pages that are made in this User Guide refers to the SAFAX User Manual [61]. Pay attention that not every single step described in [61] is repeated in this guide, but only the improvements, changes and of course the additional steps necessary in order to make use of SAFAX tool with the integration of the CAIRIS service.

### B.2.1 Advanced Functionalities

In *Chapter 3 > Advanced functionalities* (pag. 32)

In this section, all the advanced features that SAFAX offers for policy evaluation are described. The following new sub-chapter (3.8) should be added:

#### Risk Service

1. Functional Description:

The risk service to support a Risk-based access control mechanism is provided by the CAIRIS service. According to this kind of access control, the subject is authorized to perform an action over a resource only if the risk related is below a certain threshold. The idea is to implement a dynamic access control mechanism which goes beyond traditional access control systems (mostly role-based) which have quite a static approach when applying policies.

2. Caution and Warning:

Not applicable

3. Formal Description:

On the home page of SAFAX, click to create a new project, then create a new Demo. In the XACML tab upload your risk policy and your request paying attention to be coherent with the standard format that supports the different scenarios.

It is possible to have 4 different kinds of scenarios:

Scenario 1: We ask to evaluate the risk of accessing a certain resource.

For this kind of scenario the request should contain a *resource-id* attribute (which represents the asset we are asking to access in a certain mode) and of course an *action-id* attribute (which represents the mode according to which we want to access the asset, i.e. read, write,...). The policy should include, in the *Condition*, the threshold against which the risk will be compared. The UDF which has to be used in order to evaluate the condition is: *urn:bu:udf:cairis:risk:level:asset*.

Scenario 2: We ask to evaluate the risk of accessing a certain resource given a threat type.

For this kind of scenario the request should contain a *resource-id* attribute (which represents the asset we are asking to access in a certain mode), a *threat-type* attribute (which represents the threat associated to the asset) and of course an *action-id* attribute (which represents the mode according to which we want to access the asset). The policy should include, in the *Condition*, the threshold against which the risk will be compared. The UDF which has to be used in order to evaluate the condition is: *urn:bu:udf:cairis:risk:level:asset:threat:type*.

Scenario 3: We ask to evaluate the risk of accessing a certain resource in a certain context.

For this kind of scenario the request should contain a *resource-id* attribute (which represents the asset we are asking to access in a certain mode), an *environment* attribute (which represents the environment we are interested in, which is associated to the asset) and of course an *action-id* attribute (which represents the mode according to which we want to access the asset). The policy should include, in the *Condition*, the threshold against which the risk will be compared. The UDF which has to be used in order to evaluate the condition is: *urn:bu:udf:cairis:risk:level:asset:environment*.

Scenario 4: We ask to evaluate the risk of accessing a certain resource in a certain context given a threat type.

For this kind of scenario the request should contain a *resource-id* attribute (which represents the asset we are asking to access in a certain mode), a *threat-type* attribute (which represents the threat associated to the asset), an *environment* attribute (which represents the environment we are interested in, which is associated to the asset) and of course an *action-id* attribute (which represents the mode according to which we want to access the asset). The policy, should include, in the *Condition*, the threshold against which the risk will be compared. The UDF which has to be used in order to evaluate the condition is: *urn:bu:udf:cairis:risk:level:asset:threat:type:environment*.

Then go in the Risk tab (see figure B.5) and set, for the specific demo, your CAIRIS settings: credentials (Username and Password) and Database. The username and password informations correspond to the account informations that the user is supposed to possess within the CAIRIS service. The database information, instead, represents the name of the

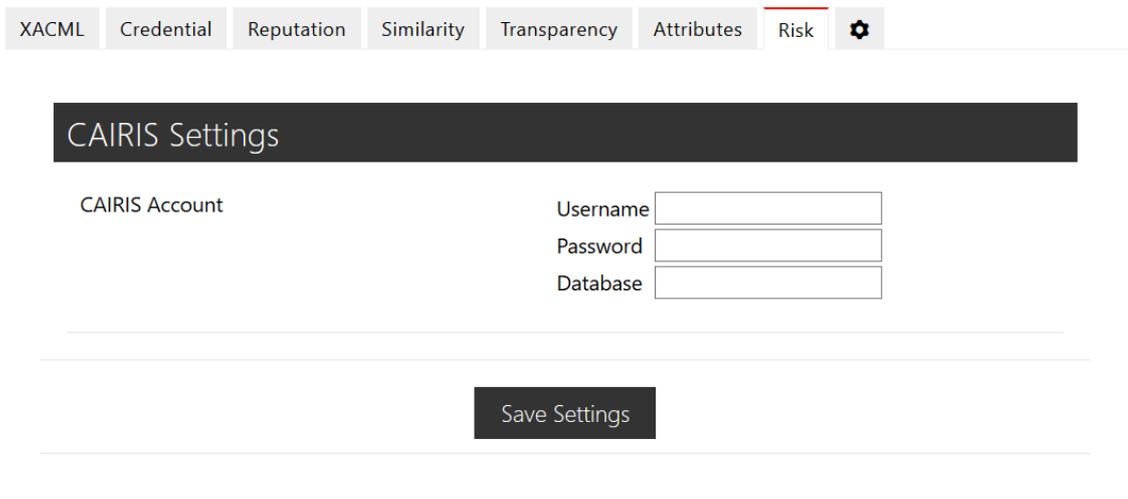
database from which the user wants to retrieve the risk scores.

Click on *Save Settings* button to effectively set the informations.

A specific set of test credentials are available, in order to let a user, without an account on CAIRIS, to test the authorization mechanism: Username = test, Password = test, Database = ACME\_Water (see figure B.6).

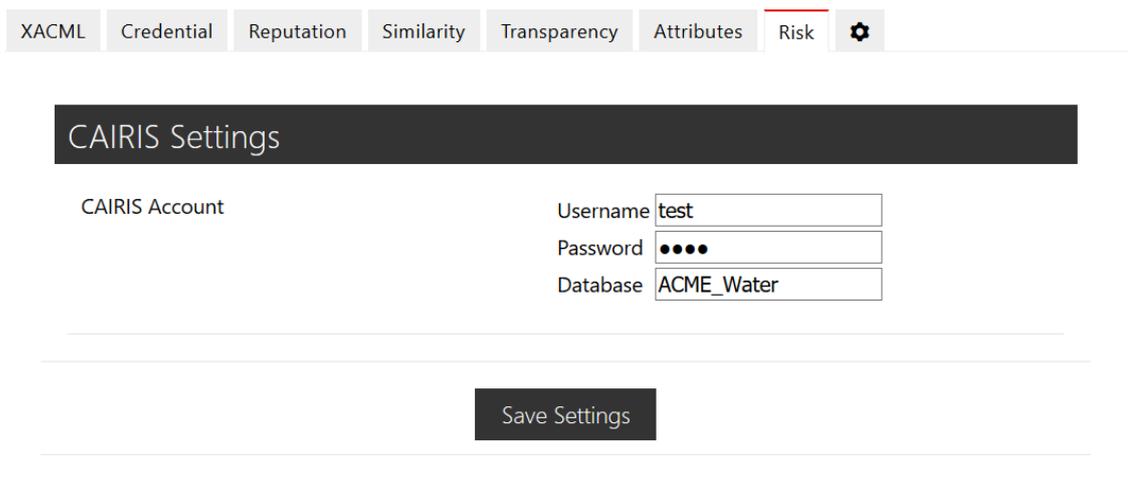
Then, after the uploads, click on run in order to evaluate your request. An example of risk response that is possible to see in the Account Activity page of SAFAX is shown in figure B.7

#### 4. Related



The screenshot shows a navigation bar with tabs: XACML, Credential, Reputation, Similarity, Transparency, Attributes, Risk, and a gear icon. The 'Risk' tab is selected. Below the navigation bar is a dark header with the text 'CAIRIS Settings'. Underneath, there is a form with the label 'CAIRIS Account'. It contains three input fields: 'Username', 'Password', and 'Database'. All three fields are currently empty. At the bottom of the form is a dark button labeled 'Save Settings'.

Figure B.5. CAIRIS Settings.



This screenshot is identical to Figure B.5, but the input fields are populated with test credentials. The 'Username' field contains the text 'test', the 'Password' field contains four black dots, and the 'Database' field contains the text 'ACME\_Water'. The 'Save Settings' button remains at the bottom.

Figure B.6. CAIRIS Settings with *test* credentials.

```
nl.tue.sec.safax.ch:> Sending Request to PDP at http://localhost/herasaf/pdp/evaluate
nl.tue.sec.safax.ch:URL in Context Handler to send to PDP web service is: http://localhost/herasaf/pdp/evaluate/503
nl.tue.sec.safax.pdp:Evaluate interface in Herasaf engine
nl.tue.sec.safax.pdp:Received Request From CH
nl.tue.sec.safax.pdp:Contacting External Trust Service at http://localhost/cairis/api/risk_level/asset/threat_type/environment
/ICT_PC/Kit_theft/Day/8/evaluate_bb48adc6fe8149b0b4d241c9405a34cf
nl.tue.sec.cairis:Cairis module in action
nl.tue.sec.cairis:CAIRIS session obtained by the user: test
nl.tue.sec.cairis:Opening CAIRIS database: ACME_Water
nl.tue.sec.cairis:
Asset: ICT PC
Threat: Kit theft
Environment: Day
Threshold: 8
Highest risk is: 9
nl.tue.sec.cairis:Risk cannot be accepted, permission has to be denied!
nl.tue.sec.safax.pdp:Received Response from External Service : {"response":"false"}
nl.tue.sec.safax.pdp:Sending Response to CH
nl.tue.sec.safax.ch:< Received Response From PDP
nl.tue.sec.safax.ch:After send request to PDP -> Sending response to FEP
nl.tue.sec.safax.pep:Received response from nl.tue.sec.safax.contexthandler.evaluate
nl.tue.sec.safax.pep:<Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:ns2="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
  <Result>
    <Decision>Deny</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

---

Figure B.7. Example of a risk response.