# Model-Driven Integration and Analysis of Access-control Policies in Multi-layer Information Systems

Salvador Martínez, Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Bouhlahia, Jordi Cabot

HAL Id: hal-01152528
https://inria.hal.science/hal-01152528

Submitted on 18 May 2015

# Model-driven Integration and Analysis of Access-control Policies in Multi-layer Information Systems

Salvador Martínez[1], Joaquin Garcia-Alfaro[2], Frédéric Cuppens[3], Nora
Cuppens-Boulahia[3] and Jordi Cabot[4]

[1] AtlanMod Team (Inria, Mines Nantes, LINA) Nantes, France
`salvador.martinez_perez@inria.fr`
[2] Télécom SudParis; RST Department CNRS Samovar UMR 5157, Evry, France
`joaquin.garcia_alfaro@telecom-sudparis.eu`
[3] Télécom Bretagne; LUSSI Department Université Européenne de Bretagne, France
`forename.surname@telecom-bretagne.eu`
[4] ICREA - UOC, Barcelona, Spain
`jcabot@uoc.edu`

**Abstract.** Security is a critical concern for any information system. Security properties such as confidentiality, integrity and availability need to be enforced in order to make systems safe. In complex environments, where information systems are composed of a number of heterogeneous subsystems, each must participate in their achievement. Therefore, security integration mechanisms are needed in order to 1) achieve the global security goal and 2) facilitate the analysis of the security status of the whole system. For the specific case of access-control, access-control policies may be found in several components (databases, networks and applications) all, supposedly, working together in order to meet the high level security property. In this work we propose an integration mechanism for access-control policies to enable the analysis of the system security. We rely on model-driven technologies and the XACML standard to achieve this goal.

## 1 Introduction

Nowadays systems are often composed of a number of interacting heterogeneous subsystems. Access-control is pervasive with respect to this architecture, so that we can find access-control enforcement in different components placed in different architectural levels, often following different AC models. However, these policies are not independent and relations exist between them, as relations exist between components situated in different architecture layers. Concretely, dependency relations exist between access-control policies, so that the decision established by rules in a policy will depend on the decisions established in another policy.

Thus, ideally, a global policy representing the access-control of the whole system should be available, as analysing a policy in isolation does not provide enough information. However, normally, this global policy only exist in an implicit and not always consistent manner. Consequently, integration mechanisms are needed in order to 1) facilitate the analysis of the security status of the whole system and 2) achieve the global security goal of the security property.

In order to tackle the aforementioned problems, we propose here a model-driven solution to integrate policies from different concrete components collaborating in an information system in a single model representation. Two requirements need to be met for achieving this goal: The use of a common access-control policy model for representing the policies of each component and the recovery/representation of the implicit dependency relations between them.

Translating all the recovered access-control policies to the same policy language, thus, representing them in a uniform way, eases the manipulation and reusability of analysis operations. In our approach, the component policies will be translated to the XACML[9] policy language while domain-specific information is added/kept by the use of profiles. Then, we complete the integration framework with a semi-automatic process for detecting the policy dependencies and to organize the policies within a single XACML model. This enables us to see the policies in our information systems as a whole. Finally, we provide a set of OCL[16] operations making use this global model and an approach to detect inter-component anomalies.

Our framework relies on the existence of high-level model representations of the policies implemented by concrete systems and on the use of model-driven tools and techniques. Consequently, as a previous step, our solution requires to perform policy recovery tasks. Concrete components often implement access-control policies by using diverse, low-level mechanisms (low level languages, database tables) to represent the rules, adding complexity to the analysis and manipulation tasks. Conversely, recovering these implemented policies and representing them in form of higher-level, more abstract models reduces the complexity and enables the reusability of a plethora of proved model-driven tools and techniques. We rely on state of the art recovery approaches [11,13,12] for this task.

The rest of the paper is organized as follows. In Section 2 we present a running example and motivation. Section 3 is devoted to the presentation of the proposed approach while in Sections 4, 5 and 6 we describe each of its steps. In Section 7 we provide details about the implementation. Finally, Section 8 discusses related work and Section 9 presents conclusions and future work.

## 2 Motivation

In order to motivate our approach, we present here an information system (IS) example that will be used through the rest of the paper.

In Figure 1, a simple, yet very common IS is depicted. This IS is composed of several components working in different architecture layers, namely, a network layer, providing networking services and enforcing access control through the firewalls (using Rule-based lists implementing Non-discretionary AC), a database layer, providing storage services and implementing role-based access-control (RBAC)[18] through its built-in permissions schema and an application level, where a Content Management System (CMS) provides publication services. This CMS also enforces RBAC by using a built-in permission schema.

As we can see, three different systems enforce access-control. These systems are not isolated but collaborate to build up the functionality of a global system that encompasses
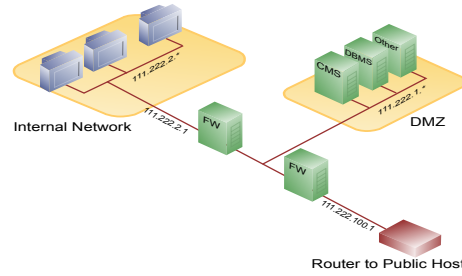
**Fig. 1.** Information System Architecture

them. Concretely, and in the case of subsystems located in different architecture layers, the collaboration relation is a dependency relation where systems in higher layers depend on services provided by lower layers. Access-control reproduces this behaviour. Consider access-control rules as functions where a decision is taken w.r.t. to a subject accessing a resource to perform a given action under certain conditions and having the following form: $R(Subject, Resource, Action, Condition) \rightarrow Decision$

Let us take a look to the following examples:

Example 1:

$R_{DB}(RoleX, TableX, Write, 8{:}00 - 16{:}00) \rightarrow accept$
$R_{FW}(Local, DBServer, Send/receive, 8{:}00 - 14{:}00) \rightarrow accept$

In this example, a given role is granted permission to access a table for modification between 8:00 and 16:00. However, the access to the database server in constrained by a firewall rule, that only allows local access to the server between 8:00 and 14:00. As the database policy depends on the firewall policy, when the latter is more restrictive, it prevails. When asking if the role can access the table under which conditions, both policies need to be taken into account in order to provide a complete answer.

Example 2:

$R_{CMS}(BlacklistedIPs, Admin, Access) \rightarrow deny$
$R_{DB}(CMSRole, CMSSchema, Write) \rightarrow accept$
$R_{FW}(0.0.0.0, DBServer, Send/receive, ) \rightarrow accept$

This example concerns the three subsystems in our IS. A rule in the CMS forbids the access to the admin pages to users located in blacklisted countries as identified by its IP address. However, the user the CMS uses to connect to the database has access for modification to the CMS database backend as stated by the second rule. Moreover, the third rule, that belongs to the firewall systems, allows to connect to the database to users in any location. Combining these three rules, a user located in a blacklisted area may be able to access the admin information on the CMS through the database backend.

From the examples, we can conclude that Access-control policies can not be regarded as isolated when they belong to systems situated in different architecture layers. Analysing the AC rules of a component for the absence of anomalies requires information from the AC policies of other components it depends on. However, this comprehensive analysis is hampered by two factors: 1) dependencies between component's
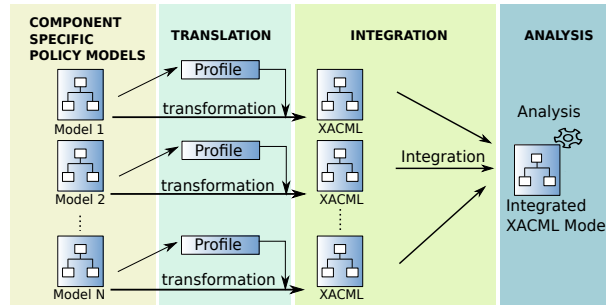
**Fig. 2.** Policy Integration Approach

policies are not explicit 2) the AC information may be represented following a different AC model and stored in different technical spaces requiring domain experts for its analysis.

## 3 Approach

In order to tackle the problems we have shown in Section 2, we propose a model-driven approach that integrates all the policies collaborating in the enforcement of access-control in a single model. Our approach requires a previous step, namely the extraction of abstract models from concrete components, and then is structured in three steps (see Figure 2):

0. **Policy recovery.** AC policies are implemented in concrete systems using a diverse set of mechanisms, often low level and proprietary, like ad-hoc rule languages, specific database dictionaries, etc. As a preliminary step for our approach we require the policies of each component to be represented in the form of abstract models, from where the complexity arising from the specificities of a given vendor or implementation technology is eliminated and only the AC information is present. This requirement is met by several previous work that investigate the recovery of access-control policies from diverse components [11,13,12]. The outputs of those works are to be the inputs of our approach.

1. **Policy Translation.** Taking as input the models described in the preliminary step, our approach proposes to translate all the policies to the same policy language. This step includes the description of extensions of the target language to make it able to represent component-specific information.

2. **Policy Integration** With all the models translated to the same language, the next step is to integrate them all in a single model, along with the dependencies between them. This step requires the discovery of such dependencies, normally implicit.

3. **Policy Analysis Support.** Having all the policies represented in the same model and the dependencies between them made explicit enables the definition of complex analysis tasks. Prior to that, the definition of a number of operations taking advantage of the model organization is required to ease the building of those analysis tasks. The third step is meant to provide that set of operations.

The following sections are devoted to a detailed description of each of the steps.

## 4  Policy Translation

All the policies in the IS, potentially conforming to different access-control models and containing domain specific information need to be translated into the same language as a previous step for the integration in a single policy. In order to do so, first, we need to chose a policy language able to represent policies following different policy models and to represent multiple policies in the same resource.

### 4.1  XACML Policy Language

XACML[9] is an access control policy language and framework fulfilling these requirements. It follows the Attribute-based access-control model (ABAC)[7] what, along with its extensibility, provides to the language enough flexibility to represent policies following different access-control models. Other approaches [15,19] describe languages and tools able to produce flexible access-control models. However, several reason inclined us to choose XACML. First of all, thanks to its ABAC philosophy, XACML is able to represent a wider range of security policies (see [7] for the capabilities of ABAC to cover other AC models), while other extensible languages like SecureUML[10] will impose the use of RBAC. Secondly, being an standard language, we expect a wider adoption and a more consistent maintenance and evolution of the language.

XACML policies are composed of three main elements *PolicySet*, *Policy* and *Rule*. A *PolicySet* can contain other *PolicySets* or a single *Policy* that is a container of *Rules* (Policy and PolicySet also specify a rule-combining algorithm, in order to solve conflicts between their contained elements). These three elements can specify a *Target* that establishes its applicability, i.e., to which combination of *Subject*, *Resource* and *Action* the *PolicySet*, *Policy* and *Rule* applies. *Subject*, *Resource* and *Action* identifies subjects accessing given resources to perform actions. These elements can hold *Attribute* elements, that represent additional characteristics (e.g., the role of the subject). Optionally, a *Rule* element can hold a *Condition* that represents a boolean condition over a subject resource or action. Upon an access request, these elements are used to get an answer of the type: permit, deny or not applicable.

### 4.2  Translation to XACML and Profiles

Our goal is to translate all the existing policies of the system in hand to XACML policies. However, the component-specific models will typically represent the access-control information in a component-specific way, i.e., they will include concepts of the domain for easing the comprehension and elaboration of policies by domain experts. Those concepts should be preserved in order to keep the expressivity of the policy. For that purpose, XACML profiles need to be defined. These profiles will basically specialize the core concepts of the XACML policy language. In general, a profile will contribute new attributes specializing the concepts of Subject, Resource and Action although specializing other concepts may be necessary mostly when the profile needs to reflects some special feature of the original policy model (take as an example the XACML RBAC Profile [5], where the concepts of PolicySet and Policy are extended as

---

[5] http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf

well as describing how to arrange these elements in a specific way to achieve the desired goal).

In order to demonstrate the process of defining a XACML profile, in the following, we describe the development of a XACML profile for the domain of relational database management systems (RDBMSs). The concepts of the domain are extracted from a security database metamodel described in [11].

First of all, note that the domain of relational databases usually relies on a RBAC model, what should be represented in the profile. There exists already a XACML profile for RBAC. Therefore, our profile will complement the use of this profile by contributing domain specific attributes for Subject, Resource and Action.

We start by defining the profile identifier that shall be used when an identifier in the form of a URI is required:

```
urn:oasis:names:tc:xacml:3.0:rdbms
```

Regarding the Resources, we will describe the following attributes.

```
urn:oasis:names:tc:xacml:3.0:rdbms:resource:database
urn:oasis:names:tc:xacml:3.0:rdbms:resource:schema
urn:oasis:names:tc:xacml:3.0:rdbms:resource:table
urn:oasis:names:tc:xacml:3.0:rdbms:resource:column
urn:oasis:names:tc:xacml:3.0:rdbms:resource:view
urn:oasis:names:tc:xacml:3.0:rdbms:resource:procedure
urn:oasis:names:tc:xacml:3.0:rdbms:resource:trigger
```

As for the actions, we will describe the following attributes, being all of type string.

```
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:insert
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:delete
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:select
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:update
urn:oasis:names:tc:xacml:3.0:rdbms:action:dbOpt:alter
urn:oasis:names:tc:xacml:3.0:rdbms:action:dbOpt:drop
urn:oasis:names:tc:xacml:3.0:rdbms:action:dbOpt:create
urn:oasis:names:tc:xacml:3.0:rdbms:action:permissionOpt:grant
urn:oasis:names:tc:xacml:3.0:rdbms:action:permissionOpt:revoke
urn:oasis:names:tc:xacml:3.0:rdbms:action:sessionOpt:set
urn:oasis:names:tc:xacml:3.0:rdbms:action:sessionOpt:connect
urn:oasis:names:tc:xacml:3.0:rdbms:action:codeOpt:execute
```

Finally, and regarding the subjects, the concept of role is already included in the RBAC profile. We will only add an attribute to identify the database elements owned by a subject, as this attribute influences the permissions (commonly, in RDBMS, the owner of a resource has all the permissions and moreover, is allowed to delegate those permissions to others).

```
urn:oasis:names:tc:xacml:3.0:rdbms:subject:owner
```

Once the profile is available, a transformation between the metamodel of the model recovered from the subsystem and the XACML (plus profiles) metamodel is defined, providing as an output XACML instance models. Note that to reflect the access-control model used in the RDBMS, we have to explicitly create a rule that in RDBMS is implicit, i.e., the owner has all the rights on the owned element.

The definition of any other profile will follow a similar process. Concretely, for the CMS we will define attributes extending the core concepts of XACML following the types defined in [13] and then combining its use with the use of the RBAC profile. As for the firewalls, several mappings to use as a basis for the profile exists, including the use of roles [4] or not [12]. We decide to extend the latter to include domain concepts (as host, zone, protocol, etc), discarding the discovery/creation of implicit roles.

## 5 Integration

Once we have all the policies represented within the same policy language, the next step is to organize the policies in a single global model. A key issue in this step is to unveil the implicit dependencies between policies situated at different architecture levels to make them explicitly appear in the model.

First of all, we need to decide the structure we will follow to represent the policies and their dependencies in a single XACML resource. The policy of each component will be stored in single XACML PolicySet, so that we can use the PolicySetIdRef to link it (without inheritance semantics) to other policies in the system. Note that some scenarios will require the policy of the component to be split in several PolicySet and Policy elements as is the case when using the RBAC profile. For simplicity, in the rest of the paper we will consider the policy of a component as the element containing its rules, disregarding how they are internally organized using XACML structural elements. Note also that the proposed structure is only intended to enable analysis capabilities and not to mimic a structure suitable, for instance, for code-generation and system deployment.

Starting from the individual policies, we need a process to discover the dependencies between them, so that the references can be properly set. We propose here a process based on exploiting context information (e.g., IP address or database-backend user for a CMS) that suggests relationships between subsystem. Note that we do not deal here with the possible heterogeneity of the properties storing the context information (different names or types) by considering that the matching of such heterogeneities may be performed, if needed, as a previous step. This context information is relevant not only to unveil the dependencies but also for the analysis of the system, thus, it needs to be stored along with the policy representation. As XACML does not provide a specific place to store this kind of information and to minimize the language extension it may require, we add this information in the description field of the PolicySet element. In this field we store a string representing a key and value map with the corresponding environment values for the Policy or PolicySet:

$$Context\{dbUserName : anonyme; IpAddress : 192.000.111.0\}$$

With the context information available, the process to find the dependencies between policies is described in Algorithm 1. Basically, for each context parameter in a given policy it searches if there is any rule using that attribute value in any of the other policies. If this is the case, a dependency exists between both policies and as such it is registered. Note that the algorithm has been optimized by considering that no circular dependencies exist. The set of candidate policies for a Policy $Pj$ (i.e., policies it may depend on), initialized to all the other policies in line 3, is modified in line 13 to remove Policies $Pi$ that already depend on it. This assumption stems from the nature of multi-layer ISs where upper components depend only on components in lower layers. This optimization can be dropped for other scenarios if needed.

Figure 3 shows the result of applying our approach to the IS example in Section 2. A *policySet* element has been created for each of the system components: firewall, database and CMS. These *policySets* contain the translated to XACML access-control *policy* of each component along with references to its dependencies as calculated by algorithm 1.

**Algorithm 1**

1:  $P \leftarrow$ All Policies
2:  **for each** $P_i \in P$ **do**
3:      $Dependency[P_i] \leftarrow \emptyset$
4:      $Candidates[P_i] \leftarrow P \backslash \{P_i\}$
5:  **end for**
6:  **for each** $P_i \in P$ **do**
7:      $C \leftarrow$ All Context Attributes in $P_i$
8:      **for each** $C_i \in C$ **do**
9:          **for each** $P_j \in Candidates[P_i]$ **do**
10:             $A \leftarrow$ All Rule Attributes in $P_j$
11:             **if** $C_i$ $in$ $A$ **then**
12:                 $Dependency[P_i] \leftarrow Dependency[P_i] \cup \{P_j\}$
13:                 $Candidates[P_j] \leftarrow Candidates[P_j] \backslash \{P_i\}$
14:             **end if**
15:         **end for**
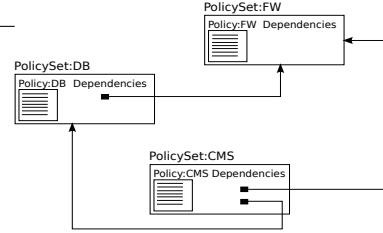16:     **end for**
17: **end for**



**Fig. 3.** Policy Organization

Considering the following set of context attributes for each component: *IpAddress : 111.222.1.10* for the database; *dbUserName:anonyme* and *IpAddress:111.222.1.12* for the CMS; the empty set for the firewall, the results is that the database *policySet* holds a dependency on the firewall *policySet* (due to the IP address context attribute) while the CMS *policySet* holds dependencies to the firewall and the database *policySets* (due to the database user and IP address context attributes).

## 6    Global analysis of inter-component anomalies

We are now able to perform all kinds of security analysis and manipulation tasks unavailable when focusing only on individual policies. The implementation of such tasks will benefit from the use of a common XACML representation which abstracts from irrelevant technical details and facilitates their reusability regardless the specific components those policies come from.

In this paper we focus on one of such analysis tasks that we believe is specially critical: the detection of inter-component anomalies. As a preparation, we will first present a number of basic operations introduced with the purpose of easing the manipulation of our integrated model (for this and other possible analysis).

### 6.1    Basic Operations

Our model can be easily queried to extract useful information by using the OCL [16] standard query language. However, there is a set of operations that will be commonly used and as such, we consider worth it to define them as a reusable library. In that sense, we present here a list of useful model manipulation operations implemented with OCL.

Table 1 presents a description of this set of basic operations. Basically, we present operations to work with the dependencies, *getDependents*, *getDependencies*, *getAllDependencies resolveDependency* and *getDependencySource*; operations to obtain the context attributes of a policy, *getContextAttributes*; and operations to obtain the rules related with context attributes in a dependency relation, *getRelevantRules*.

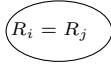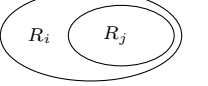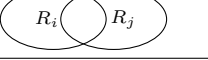| Operation | Description |
|---|---|
| getDependents(p:Policy) : Sequence{Policy} | Given a policy P, returns the sequence of policies having this policy as dependency. |
| getDependencies(p:Policy) : Sequence{Dependency} | Given a policy P, returns the sequence of direct dependencies. |
| getAllDependencies(p:Policy): Sequence{Dependency} | Given a policy P, returns the sequence of ALL the dependencies, direct and indirect. |
| resolveDependency(d:Dependency) : Policy | Given a dependency D, returns its target Policy P. |
| getDependencySource(d:Dependency): Policy | Given a dependency D, returns its source Policy P. |
| getContextAttributes(p:Policy) : Sequence{Tuple{key:String,val:String}} | Given a Policy P, returns a sequence of tuples{key:String,Value:String}, representing the context attributes |
| getRelevantRules(p:Policy,p2:Policy) : Sequence{Rule} | Given two policies, $Pi$ and $Pj$, with $Pi$ dependent on $Pj$, returns the rules in $Pj$ related to the context attributes of $Pi$, i.e., the set of rules of $Pj$ $Pi$ depends on |

**Table 1.** OCL Operations

## 6.2 Detection of rule anomalies

One important analysis task is the detection of anomalies that appear when several policies work together, as shown in the examples in Section 2. The problems these anomalies cause vary from simply increasing the complexity of policies to the introduction of unexpected behaviour of a component w.r.t. its defined policy.

Focusing on the undesired effects these anomalies may produce and considering rule $r_i$ depending on $r_j$ (as indicated by the policy dependency structure presented in 5), we identify the following risks (defining risk as a threat caused by an anomaly that may lead to loses in terms of money and/or reputation):

- **Security Risk**: The combination of $r_i$ and $r_j$ may cause a security hole. This happens when $r_j$ allows requests for values $r_i$ does not allow. We consider the risk partial when $r_j$ only allows some of the $r_i$ denied values. Example 2 in Section 2 belongs to this category, as the network layer, combined with the database layer, allows request the CMS does not.
- **Service Risk**: The combination of $r_i$ and $r_j$ may cause the component to which $r_i$ provides access-control not to be able to provide the expected service. This happens when $r_j$ denies requests for values $r_i$ allows. We consider the risk partial when $r_j$ only denies some of the $r_i$ allowed values. Example 1 in Section 2 shows a partial service risk.
- **Redundancy**: $r_i$ or $r_j$ may be eliminated without impact to the behaviour in the system. This may happen when both rules deny requests for the same values. Policies containing those rules may be refactored to reduce complexity.
- **No Risk**: The combination of $r_i$ and $r_j$ does not generate any risk.

As we have seen, these anomalies depend on the relations that hold between the set of request matched by a pair of rules. For determining that relation we need to compare security rules. This comparison is done for the purpose of checking if 1) Conditions in different rules hold for the same set of values 2) The rule effect when the conditions hold are conflicting. This process, which we call rule similarity evaluation following the terminology in [14], can be performed following different approaches. Here, due to the relative simplicity of the syntactical analysis they propose, we adapt the approach proposed in [14] to the case of policies in different architectural layers. Other approaches could however be also adapted to our specific case.

| Rule similarity type | Matched requests | $R_i^{Accept}, R_j^{Accept}$ | $R_i^{Deny}, R_j^{Deny}$ | $R_i^{Deny}, R_j^{Accept}$ | $R_i^{Accept}, R_j^{Deny}$ |
|---|---|---|---|---|---|
| $Ri\ Converges\ Rj$ | $R_i = R_j$ | No Risk | Redundancy | Security Risk | Service Risk |
| $Ri\ Diverges\ Rj$ | $R_i$  $R_j$ | Service Risk | No Risk | No Risk | No Risk |
| $Ri\ Restricts\ Rj$ | $R_j$  $R_i$ | No Risk | No Risk | Security Risk | Service Risk |
| $Ri\ Extends\ Rj$ | $R_i$  $R_j$ | Service Risk* | No Risk | Security Risk* | Service Risk* |
| $Ri\ Shuffles\ Rj$ | $R_i$  $R_j$ | Service Risk* | No Risk | Security Risk* | Service Risk* |

\* partial risk

**Table 2.** Policy rule similarity type instantiated

**Rule Similarity.** For risk analysis purposes, the similarity of rules can be classified in five values {*Converges, Diverges, Restricts, Extends, Shuffles*} with the following definition (see the second column in Table2 for a graphical representation):

*Converge*: Two rules 'converge' if the sets of values are equal with respect to which their conditions hold.

*Diverge*: Two rules 'diverge' if the sets of values do not intersect with respect to which of their conditions hold.

*Restrict and extend*: A rule 'restricts' (or 'extends') another rule if the sets of values with respect to which its conditions hold contain (or is contained in) the set of values computed for the other rule.

*Shuffle*: Two rules 'shuffle' if the sets of values for which their conditions hold intersect, but no one is contained in the other.

These values are calculated by a similarity calculation algorithm presented in [14]. We instantiate the Rule similarity types for the case of rules situated in different architectural layers by assigning them the previously defined risk types.

**Risk calculation.** The assignment of risk types to rule similarity types depends on their effect (deny, accept) and matched request sets. Table 2 shows the assignment for all the possible combinations. Notice that the actual presence of anomalies between two rules depends on the nature of the involved systems and how they interact.

Algorithm 2 describes the process of instantiating the risks of rules over our infrastructure given a rule and an attribute to check. Basically, the algorithm iterates over the policies the policy containing the rule depends on, retrieving relevant rules (lines 9 and 13) and retrieving the similarity value (line 18) to produce an anomaly report (line 19). It is important to note that when the dependency is indirect, i.e., the dependency relationship is established through another policy, we need to get the relevant rules w.r.t. this latter policy having the direct dependency (line 11 to 15). This is specially important because a given policy may have both, a direct and an indirect dependency with

---

**Algorithm 2** Risk evaluation

---

1: $r \leftarrow Initial\,rule, a \leftarrow Initial\,attribute$
2: $P \leftarrow P/r \in P, D \leftarrow getAllDependencies(P), S \leftarrow getDependencies(P)$
3: **for each** $D_i \in D$ **do**
4:     $P_i \leftarrow resolveDependency(D_i)$
5:     **if** $D_i$ $in$ $S$ **then**
6:         $R \leftarrow getRelevantRules(P, P_i)$
7:         $tagRules(R, P)$
8:     **else**
9:         $P_j \leftarrow getDependencySource(D_i)$
10:        $R \leftarrow getRelevantRules(P_j, P_i)$
11:        $tagRules(R, P_j)$
12:     **end if**
13:     **for each** $r_i \in R$ **do**
14:        **if** $a \in r_i$ **then**
15:           $sim \leftarrow calculateSimilarity(r_i, r, a)$
16:           $reportAnomalyCheck(sim, r_i, r, )$
17:        **end if**
18:     **end for**
19: **end for**

---

another policy, each one yielding a different set of relevant rules. As this information is relevant for performing further analysis, each rule is tagged with its dependent policy (lines 10 and 14).

Let us take a look of how the risk types instantiation is calculated for the examples presented in Section 2. Regarding the first example, we want to know if given the database rule $R_{DB}$ and its time attribute there exists an anomaly:

$R_{DB}(RoleX, TableX, Write, 8{:}00 - 16{:}00) \rightarrow accept$

Following the proposed algorithm, the policy dependencies are retrieved, that in this case consists only in a dependency towards the firewall policy. Using the context attributes, the rules in the firewall policy related to the database are retrieved. Finally, from this set of rules, the ones containing the time attribute are checked for similarity with the database rule and tagged in consequence. We can then show only those having a similarity implying an anomaly. In that subset we will have the second rule in the example, $R_{FW}$, as it uses a context attribute, the time attribute, and the calculated similarity has the value of extend, which may cause an anomaly of partial service risk, as shown in the Table 2.

$R_{FW}(Local, DBServer, Send/receive, 8{:}00 - 14{:}00) \rightarrow accept$

As for the second example, the process starts in a similar way, by retrieving the dependencies of the CMS policy containing the rule $R_{CMS}$ and the attribute to be checked, in this case, the source IP address.

$R_{CMS}(BlacklistedIPs, Admin, Access) \rightarrow deny$

However, now we will have two kinds of dependencies. The CMS policy depends directly on the database and firewall policies, but it also holds an indirect dependency to the firewall policy through the database one. Thus, three sets of rules are retrieved, those of the firewall and database policies related to the CMS context attributes (IP address and database user) and those of the firewall related with the context attributes of the database (IP address of the server).

The risk type instantiation calculated on the set of retrieved rules will exhibit not only the possible anomalies the policy of the CMS has with respect to the database and the firewall directly, but also those anomalies arising from the combination of the effects of rules in the database and firewall together. Thus, among other possible anomalies present in the firewall or database configuration we will retrieve the one associated with the following rule:

$$R_{FW}(0.0.0.0, DBServer, Send/receive, ) \rightarrow accept$$

It gives access to the database server (back-end of the CMS) to users in a location forbidden by the CMS policy. This rule retrieved from the database dependency and tagged that way, informs us about an anomaly (shadowing) between the CMS and the firewall involving the database system. The security expert will only need to retrieve the database relevant rules w.r.t. the CMS policy to have a complete picture of the problem.

$$R_{CMS}(BlacklistedIPs, Admin, Access) \rightarrow deny$$
$$R_{DB}(CMSRole, CMSSchema, Write) \rightarrow accept$$
$$R_{FW}(0.0.0.0, DBServer, Send/receive, ) \rightarrow accept$$

Obtaining this information would not have been possible without the integration of the policies and the discovery of their dependencies.

## 7 Implementation

In order to validate the feasibility of our approach, a proof-of-concept prototype implementation has been developed under the Eclipse environment[6] by using Model-driven tools and techniques. Concretely, our implementation is based on two features:

**Model Representation.** Our approach takes as input domain-specific access-control models extracted from different components in order to translate them to XACML models. To be able to do that, a XACML policy metamodel (models conform to metamodels, which define the main concepts and relationships of the domain) is required, so that models conforming to it can be created. We have used, EMF, the de-facto modeling framework for that purpose.

Providing the XACML XSD schema[7] as an input to EMF, the framework allowed us to generate the XACML policy metamodel, and in turn, to generate Java code plugins for the manipulation of model instances, including a tree-based editor. Note that these models instances can be, in turn, serialized using a XML syntax. XACML identifiers, datatypes, etc, are integrated in a similar way i.e., by extracting a metamodel through EMF and linking it to the XACML metamodel.

**Model Query&Transformations.** Once the means to represent XACML models are available, we can perform the transformations from the domain models and the operations and algorithms described in Sections 5 and 6. We have used the ATL[8] model-to-model transformation language for that purpose. ATL is a hybrid (declarative with

---

[6] https://www.eclipse.org/

[7] http://docs.oasis-open.org/xacml/3.0/XSD/cs-xacml-schema-policy-01.xsd

imperative facilities) language and framework that provides the means to easily specify the way to produce target models from source models. The following model-to-model transformation have been created:

1) A model transformation for each component model to a XACML model.

2) A library of helpers, an ATL mechanism to factorize OCL operations, representing the basic operations in section 6

3) A model transformation for the integration algorithm in 1.

4) A model query for the detection of anomalies, following the algorithm 2.

## 8   Related Work

The integration of security policies is a research problem that has attired the attention of the security research community in the recent years. Consequently, different approaches to tackle the problem have been proposed.

From a formal perspective, in [3] the authors provide the foundations of a formal framework to represent policies in different architectural layers. Similarly, in [2] the authors analyze different combination operations for AC policies. Among them, the combination of heterogeneous policies and the integration of hierarchical policies through refinement. [1] provides a logical framework to encode multiple authorization policies into a proof-carrying authorization formalism. In [17] Method-B is used to formalize the deployment of AC policies on systems composed of several (network) components. Finally, by using model-driven techniques, in [5] the authors formalize the policy continuum model, that represent policies at different inter-related abstraction layers although it does not tackle the problem of inter-related architectural layers.

The aforementioned works are valuable contributions that could be eventually used to enforce a forward engineering process to generate correct policies. However, none of these formalization works provide the bridges necessary to fill the gap between real deployed policies and the proposed formalisms as they mostly aim at providing a formal framework to deploy/analyse/manipulate synthetic policies. Conversely, our approach works the other way round by proposing a more pragmatic approach, aimed at providing a solution for the integration of real, already deployed policies.

More similar to us and working on XACML policies, in [6] the authors describe approaches to detect anomalies while in [14] integration analysis for policies belonging to different authorization entities is proposed. However, they do not deal with dependencies between policies. Finally, we have adapted the similarity process in [14] to compare rules and policies for the case of inter-dependent access control policies.

## 9   Conclusions and Future Work

We have presented an approach to integrate the Access-control policies collaborating in an Information System. It translates all the policies to the XACML policy language and organizes them in an unique model by unveiling the implicit dependencies between them. Finally, we have presented useful operations taking advantage of the proposed infrastructure that lead to detect possible anomalies between the policies.

As a future work, we plan to extend our approach to include other sources of information. Concretely, we would like to integrate in our approach the information provided by the audit and logging systems of IS components. So far, we can point the security experts to possible anomalies. By analysing the audits together with them, we believe we can determine whether the anomaly is taking place/being exploited or not. Finally, we also intend to extend our approach to integrate different kinds of policies. Privacy, Integrity and Secrecy policies may collaborate between them and thus, we believe they may benefit of an integration approach as the one we have presented here.

# References

1. L. Bauer and A. W. Appel. *Access Control for the Web via Proof-Carrying Authorization*. PhD thesis, Princeton University, 2003.
2. P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An Algebra for Composing Access Control Policies. *TISSEC*, 5(1):1–35, 2002.
3. M. M. Casalino and R. Thion. Refactoring Multi-Layered Access Control Policies Through (De)composition. In *CNSM*, pages 243–250, 2013.
4. F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A Formal Approach to Specify and Deploy a Network Security Policy. In *FAST'04*, pages 203–218, 2004.
5. S. Davy, B. Jennings, and J. Strassner. The Policy Continuum–Policy Authoring and Conflict Analysis. *Computer Communications*, 31(13):2981–2995, 2008.
6. H. Hu, G.-J. Ahn, and K. Kulkarni. Anomaly Discovery and Resolution in Web Access Control Policies. In *SACMAT'11*, pages 165–174. ACM, 2011.
7. X. Jin, R. Krishnan, and R. Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, pages 41–55. Springer, 2012.
8. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A Model Transformation Tool. *Science of Computer Programming*, 72(1):31–39, 2008.
9. H. Lockhart, B. Parducci, and A. Anderson. OASIS XACML TC, 2013.
10. T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *UML'02*, pages 426–441. Springer, 2002.
11. S. Martínez, V. Cosentino, J. Cabot, and F. Cuppens. Reverse Engineering of Database Security Policies. In *DEXA (volume 2)*, pages 442–449, 2013.
12. S. Martínez, J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and J. Cabot. Model-Driven Extraction and Analysis of Network Security Policies. In *MoDELS*, pages 52–68, 2013.
13. S. Martínez, J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and J. Cabot. Towards an Access-Control Metamodel for Web Content Management Systems. In *ICWE Workshops*, pages 148–155, 2013.
14. P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. XACML Policy Integration Algorithms. *TISSEC*, 11(1):4, 2008.
15. T. Mouelhi, F. Fleurey, B. Baudry, and Y. L. Traon. A Model-Based Framework for Security Policy Specification, Deployment and Testing. In *MoDELS 2008*, pages 537–552, 2008.
16. OMG. *OCL, version 2.0*. Object Management Group, June 2005.
17. S. Preda, N. Cuppens-Boulahia, F. Cuppens, J. García-Alfaro, and L. Toutain. Model-Driven Security Policy Deployment: Property Oriented approach. In *ESSoS*, pages 123–139, 2010.
18. R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST Model for Role-Based Access Control: Towards A Unified Standard. RBAC '00, pages 47–63. ACM, 2000.
19. B. Trninic, G. Sladic, G. Milosavljevic, B. Milosavljevic, and Z. Konjovic. PolicyDSL: Towards Generic Access Control Management Based on a Policy Metamodel. In *SoMeT*, pages 217–223, 2013.