Refactoring access control policies for performance improvement

Donia El Kateb
University of Luxembourg
6 rue Coudenhove-Kalergi
L-1359 Luxembourg
donia.elkateb@uni.lu

Tejeddine Mouelhi
University of Luxembourg
6 rue Coudenhove-Kalergi
L-1359 Luxembourg
tejeddine.mouelhi@uni.lu

Yves Le Traon
University of Luxembourg
6 rue Coudenhove-Kalergi
L-1359 Luxembourg
yves.letraon@uni.lu

JeeHyun Hwang
Dept. of Computer Science,
North Carolina State
University
Raleigh, NC 27695, U.S.A
jhwang4@ncsu.edu

Tao Xie
Dept. of Computer Science,
North Carolina State
University
Raleigh, NC 27695, U.S.A
xie@csc.ncsu.edu

ABSTRACT

Modern access control architectures tend to separate the business logic from access control policy specification for the sake of easing authorization manageability. Thus, request evaluation is processed by a Policy Decision Point (PDP) that encapsulates the access control policy and interacts with the business logic through Policy Enforcement Points (PEPs). Such architectures may engender a performance bottleneck due to the number of rules that have to be evaluated by a single PDP at decision making time. In this paper, we propose to optimize the decision-making process by splitting the PDP into smaller decision points. We conducted studies on XACML (eXtensible Access Control Markup Language) to identify the best PDP splitting configuration. Our evaluation results show that the best splitting criterion is the one that preserves the initial architectural model in terms of interaction between the business logic and the decision engine and enables to reduce the time of request evaluation time by up to 9 times.

Keywords

Performance, Optimization, Access Control Policies, PEP, PDP, XACML

1. INTRODUCTION

Access control policies are derived from an access control model like RBAC, MAC, DAC and OrBAC [8, 9, 10, 12] to define authorization strategies which regulate the actions that users could perform on system resources. In the context of a Policy based software systems, access control architectures are often built with respect to a popular architectural concept, which separates a Policy Enforcement Point (PEP) from a Policy Decision Point (PDP) [7]. More specifically, the PEP is located inside the application (i.e., business logic of the system) and formulates requests. The PDP encapsulates the policy. The PDP then receives requests from the PEP and returns evaluation responses (e.g., permit or deny) by evaluating these requests against rules in the policy. This architecture enables to facilitate managing access rights in a fine-grained way since it decouples the business logic from the access control decision logic, which can be standardized.

In this architecture, the policy is centralized in one point, the PDP. Centralization of the PDP offers advantages such as facilitating managing and maintaining the policy, which is encapsulated in a single PDP. However, such centralization can be a pitfall for degrading performance since the PDP manages all of access rights to be evaluated by requests issued by PEPs. Moreover, the dynamic behaviour of organization's business, the complexity of its workflow and the continuous growth of its assets may increase the number of rules in the policy [3]. This fact may dramatically degrade the decision-making time, lead to performance bottlenecks and impact service availability.

In this paper, we tackle perfomance requirements in policy based software systems and propose a mechanism for refactoring access control policies to reduce the request evaluation time. Reasoning about performance requirements in access control systems is intended to enhance the reliability and the efficency of the overall system and to prevent business losses that can be engendered from slow decision making. In our approach, we consider policy-based software systems, which enforce access control policies specified in the eXtensible Access Control Markup Language (XACML) [2]. XACML is a popularly used XML-based language to specify rules in a policy. A rule specifies which subjects can take which actions on resources in which condition.

We propose an automated approach for optimizing the process of decision making in XACML request evaluation. Our approach includes two facets: (1) performance optimization criteria to split the PDP into smaller decision points, (2)

architectural property preservation to keep an architectural model in which only a single PDP is triggered by a given PEP. The performance optimization facet of the approach relies in transforming the single PDP into smaller PDPs, through an automated refactoring of a global access control policy (i.e., a policy governing all of access rights in the system). This refactoring involves grouping rules in the global policy into several subsets based on splitting criteria. More specifically, we propose a set of splitting criteria to refactor the global policy into smaller policies. A splitting criterion selects and groups the access rules of the overall PDP into specific PDPs. Each criterion-specific PDP encapsulates a sub-policy that represents a set of rules that share a combination of attribute elements (Subject, Action, and/or Resource). Our approach takes as input a splitting criterion and an original global policy, and returns a set of corresponding sub-policies. Given a set of requests, we then compare evaluation time of the requests against the original policy and a set of sub-polices based on the proposed splitting criteria.

The second facet aims at keeping the cardinality based feature between the PEP and the PDP and by selecting the splitting criteria that do not alter the initial access control architecture. The best splitting criteria must thus both improve performance and comply to our architecture: the relationship must be maintained, linking each PEP to a given PDP. When refactoring the system, each PEP in our system can be mapped to a set of rules that will be relevant when evaluating requests. We automate this refactoring process and conduct an evaluation to show that our approach enables the decision-making process to be 9 times faster than using only a single global access control policy.

In our previous work [5], we addressed performance issues in policy evaluation by using a decision engine called XEngine that outperforms Sun PDP implementation [1]. XEngine transforms an original XACML policy into its corresponding policy in a tree format by mapping attribute values with numerical values. Our contribution in this paper goes in the same research direction as it aims to reduce the request evaluation time by refactoring the policy. Our evaluation results show that evaluation time is reduced up to nine times with split policies if compared to its original global policy and that we have a considerable performance improvement, if the policies resulting from our refactoring process are evaluated with XEngine rather than Sun PDP.

This paper makes the following three main contributions:

- We develop an approach that refactors a single global policy into smaller policies to improve performance in terms of policy evaluation.
- We propose a set of splitting criteria to help refactor a policy in a systematic way. The best splitting criterion is the one that does not alter the initial access control architecture.
- We conduct evaluations on real-life policies to measure performance improvement using our approach. We compare our approach with an approach based on the global policy in terms of efficiency. Our approach achieves more than 9 times faster than that of the approach based on the global policy in terms of evaluation time.

The remainder of this paper is organized as follows: Section 2 introduces flow models for access control policy in XACML, and access policy architecture model. Section 3 presents our adopted strategy and Section **??** presents evaluation results and discusses the effectiveness of our approach. Section **??** discusses related work. Section **??** concludes this paper and discuses future research directions..

2. CONTEXT/PROBLEM STATEMENT

The present section clarifies the motivation for the work presented in this paper and introduces the synergy requirement that we aim to maintain while reasoning about performance requirements for access control policies.

2.1 Synergy requirement related to the considered Access Control Architecture

Most access control scenarios involve an access control policy which is modeled, analyzed and implemented as a separate component encapsulated in a PDP. Figure 1 illustrates the interactions between the PEPs and the PDP: the PEP calls the PDP to retrieve an authorization decision based on the PDP encapsulated policy. This authorization decision is made through the evaluation of rules in the policy. Subsequently, an authorization decision (permit/deny) is returned to the PEP. The separation between the PEP and the PDP in access control systems simplifies policy management across many heterogeneous systems and enables to avoid potential risks arising from incorrect policy implementation, when the policy is hardcoded inside the business logic.

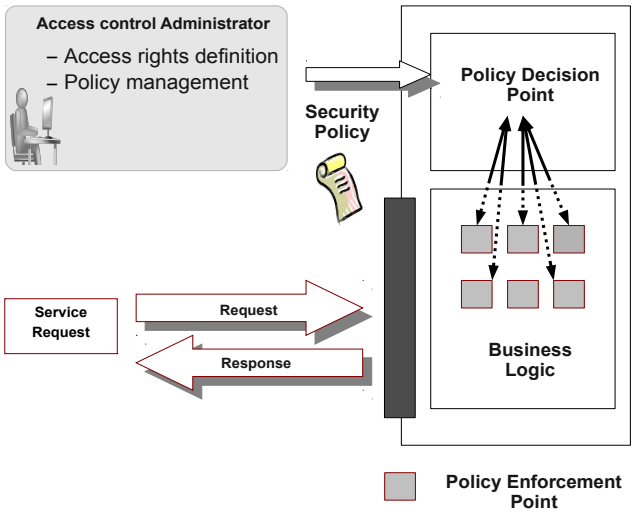

Figure 1: Access Control Request Processing

Managing access control policies is one of the most challenging issues faced by organizations. Frequent changes in access control systems may be required to meet business

needs. An access control system has to handle some specific requirements like role swap when employees are given temporary assignments, changes in the policies and procedures, new assets, users and job positions in the organization. All these facts make access control architectures very difficult to manage, and plead in favor of a simple access control architecture that can easily handle changes in access control systems. Along with the reasoning about performance, we propose to maintain the simplicity of the access control architecture whose model is presented in Figure 2. In this model, a set of business processes, which comply to users' needs, are encapsulated in a given business logic which is enforced by multiple PEPs. Conceptually, the decision is decoupled from the enforcement and involves a decision making process in which each PEP interacts with one single PDP, thus a single XACML policy is evaluated to provide the suitable response for an access control request provided by a given PEP.

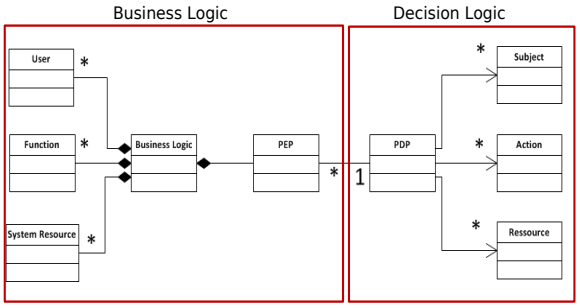

Figure 2: The Access Control Model

Considering a single XACML policy file for each initiating PEP enables to ease policies management and to maintain a simple architecture in which a given PEP is mapped to a fixed PDP at each decision making process. We define the synergy requirement in the access control architecture by specialization of access control policies to be used for a particular PEP before deployment. The goal behind maintaing a single policy related to each PEP in the system is to keep a strong traceability between what has been specified by the policy and the internal security mechanisms enforcing this policy at the business logic level. In such setting, when the access control policy rules are updated or removed, the related PEPs can be easily located and removed and thus the application is updated in a synchronous way with the policy changes.

2.2 XACML Access Control Policies and Performance Issues

In this paper, we focus on access control decision making for policies expressed in XACML. XACML [2] is a standard policy specification language that defines a syntax of access control policies in XML. In the XACML paradigm, a policy set is a sequence of policies, a combining algorithm and a target. A policy element is expressed through a target, a set of rules and a rule combining algorithm. The target defines the set of resources, subjects and actions to which a rule is applicable. The rule element consists of a target, a condition, and an effect. The condition is a boolean expression that specifies restrictions (e.g., time and location restrictions) on the elements in the target. It represents the environmental context in which the rule applies. Finally, the effect element is either permit or deny. When more than one rule is applicable to a given request, the combining algorithm enables selecting which rule to choose. For example, given two rules, which are applicable to the same request and provide different decisions, the permit-overrides algorithm prioritizes a permit decision over the other decision. More precisely, when using the permit-overrides algorithm, the policy evaluation engenders the following three decisions:

- Permit if at least one permit rule is applicable for a request.
- Deny if no permit rule is applicable and at least one deny rule is applicable for a request.
- NotApplicable if no rule is applicable for a request.

Figure 3 shows a simple XACML policy that denies subject Bob to borrow a book.

```
<PolicyId="XACML Policy Example"                                  } Combining Algorithm
 RuleCombiningAlgId="combining-algorithm:permit-overrides">
<Target>                                                          }
 <Subjects> <AnySubject/> </Subjects>                             } Empty Policy Target
  <Resources> <AnyResource/> </Resources>                         }
  <Actions> <AnyAction/> </Actions>                               }
</Target>

<Policy>
<Rule RuleId="R1" Effect="Deny">
   <Target>
    <Subjects>
     <Subject>                                                    }
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">  } Subject
       <AttributeValue>Bob</AttributeValue>                       }
      </SubjectMatch>                                             }
     </Subject>                                                   }
    </Subjects>
    <Resources>                                                   }
     <Resource>                                                   } Resource
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">  }
       <AttributeValue>Book</AttributeValue>                      }
      </ResourceMatch>                                            }
     </Resource>                                                  }
    </Resources>
    <Actions>                                                     }
     <Action>                                                     } Action
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">  }
       <AttributeValue>Borrow</AttributeValue>                    }
      </ActionMatch>                                              }
     </Action>                                                    }
    </Actions>
   </Target>
  </Rule>
</Policy>
```

Figure 3: XACML Policy Example

The XACML request encapsulates attributes that define which subject is requested to take an action on which protected resource in a system. This can be under/not a condition. A request, which satisfies both target and condition in a rule, is evaluated to decision specified in the rule's effect element. If the request does not satisfy target and condition elements in any rule, its response yields "NotApplicable" decision.

XACML enables administrators to externalize access control policies for the sake of interoperability since access control policies can be designed independently from the underlying programming language or platform. Such flexibility enables to easily update the access control policy to support new requirements.

However, the increasing complexity of organizations in term of structure, relationships, activities, and access control requirements has impacted access control policy design. This has led to deal with access control policies where the number of rules depicting the associations between resources, users and actions is in continuous increase. Moreover, the policy is usually centralized in a single point PDP, which has to manage multiple access control requests.

These observations raise performance concerns related to request evaluation time for XACML access control policies. In fact, performance bottlenecks degrade the system efficiency which slows down the overall business processes. Considering XACML request evaluation, many factors may lead to reducing XACML requests evaluation performance.
Next, we present some of the factors that are causing this performance issue in XACML architecture:

- An XACML policy may contain several attributes that constitute descriptive information about the target elements, the retrieval of varying size of attributes values for each request within XML elements in a request, may increase the evaluation time.
- A PolicySet includes a set of policies, the effect of the whole PolicySet is determined by combining the effects of the policies according to a combining algorithm, the computation of resulting effect from policies is also considered as a potential evaluation time latency.
- Conditions evaluation in XACML rules may slow down the decision making process since these conditions can be complex because they can be built from an arbitrary nesting of non-boolean functions and attributes.

3. APPROACH

This section describes our approach and gives an insight of the process supporting it. In the first step, we show how performance improvement can be achieved by reducing the number of access control rules that have to be considered at the evaluation time, through the definition of policy based splitting criteria. Secondly, we show how to select the splitting criterion that preserves the synergy requirement in the access control architecture.

3.1 XACML Policy Refactoring Process

What happens in a request evaluation process is that for a given access control request, some rules are evaluated in the global policy whereas some of those rules are not applicable to the request. Starting from this observation, we propose to evaluate only the relevant rules for a given request in the decision making process. We propose to split the initial policy into smaller policies based on attribute values combination. We transform the policy P into smaller policies P_{SC_w} where each policy conforms to a Splitting Criteria SC_w. An SC_w defines the set of attributes that are considered to classify all the rules into subsets having the same attribute values. w denotes the number of attributes that have to be considered conjointly for aggregating rules into a unified set based on specific attribute elements selection. A policy can be split into a set of policies where the rules have the same subject, resource, or action. Rules can also be aggregated considering two attributes like $< Subject, Action >$ or $< Action, Resource >$, in this setting, a policy will be transformed into a set of smaller policies where rules in the resulting policies have the same couple of attribute elements. We can go further in our grouping strategy when using P_{SC_w} whose rules match a specific triplet of resource, action and subject. Table I shows all splitting criteria categories according to the attribute elements combination.

Categories	Splitting Criteria
SC_1	$< Subject >, < Resource >, < Action >$
SC_2	$< Subject, Action >, < Subject, Resource >$ $< Resource, Action >$
SC_3	$< Subject, Resource, Action >$

Table 1: Splitting Criteria

Once the splitting process is performed, the access control architecture will include one or more (PDPs) that comply with a certain splitting criterion.

3.2 Architecture Model Preservation: PEP-PDP Synergy

We consider the different splitting criteria that we have identified in the previous section and we propose to select the splitting criterion that enables to preserve the synergy requirement in the access control architecture. This splitting criterion respects how PEPs are organized at the application level and how they are linked to their corresponding PDPs. A deep analysis of the PEPs at the application enables to observe the mapping between the PEPs and the PDP. At the application level, the PEP is represented by a method call that triggers a decision making process by activating some specific rules in the PDP. The code below is taken from [**?**], this code excerpt shows an example of a PEP represented by the method checkSecurity which calls the class SecurityPolicyService that initiates the PDP component.

```
public void borrowBook(User user, Book book)
throws SecuritPolicyViolationException
// call to the security service
    ServiceUtils.checkSecurity(user,
LibrarySecurityModel.BORROWBOOK_METHOD,
LibrarySecurityModel.BOOK_VIEW);
ContextManager.getTemporalContext());
// call to business objects
// borrow the book for the user
    book.execute(Book.BORROW, user);
// call the dao class to update the DB
    bookDAO.insertBorrow(userDTO, bookDTO);
```

An analysis of this code reflects that this PEP will trigger exclusively all the rules organized by the couple (Action, Resource) for the subject user given as input parameter. This is due to the call of the following methods: LibrarySecurityModel.BORROWBOOK_METHOD and LibrarySecurityModel.BOOK_VIEW. Thus the splitting process that will

preserve the mapping between the PEPs and the PDP will be $SC_2 =< Resource, Action >$ in this case. Depending on the application, establishing this mapping may require to identify all the enforcements points in an application, and to track the different method calls triggered from these specific enforcement calls to map them to the relevant access control rules.

In the worse case, splitting the initial PDP into multi-PDPs may lead to a non-synergic system: a PEP may send its requests to several PDPs. The PDP, which receives a request is only known at runtime. Such a resulting architecture breaks the PEP-PDP synergy and the conceptual simplicity of the initial architecture model. In the best case, the refactoring preserves the simplicity of the initial architecture, by keeping a many-to-one association from PEPs to PDPs. A given request evaluation triggered by one PEP will always be handled by the same PDP. Operationally, the request evaluation process will involve one XACML policy file. In this case, the refactoring is valid, since its does not impact the conceptual architecture of the system.

Our empirical results, presented in section **??**, have shown that adopting a policy refactoring based on system functions, as a refactoring strategy, enables to have the best splitting criterion.

As depicted in Figure **??**, the refactoring process is automated and starts by specifying and creating the XACML file which will be split by our tool according to a specified SC that can be chosen by an access control stakeholder. Afterwards, the policies are included in the framework that supports our approach. For every change in the access control policy, the initial policy is updated and split again in order to be included again in the framework. From a point of view of the system administration, maintaining and updating the access control policies is completely a standalone and simple process. The input is usually a centralized XACML policy. This input remains the same than before the access control performance issue is tackled. Our process is transparent in the sense that it does not impact the existing functional aspects of the access control management system, for system administrators, who have to update the policy fequently and have to manage various dimensions of access control systems such the scalability and maintainability.

4. REFERENCES

[1] OASIS eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/xacml/, 2005.

[2] Sun's XACML implementation. http://sunxacml.sourceforge.net/, 2005.

[3] J. Chaudhry, T. Palpanas, P. Andritsos, and A. Mana. Entity lifecycle management for okkam. In *Proc. 1st IRSW2008 International Workshop on Tenerife*, 2008.

[4] A. D. Keromytis and J. M. Smith. Requirements for scalable access control and security management architectures. *ACM Trans. Internet Technol.*, 7, May 2007.

[5] A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: A fast and scalable XACML policy evaluation engine. In *Proc. International Conference on Measurement and Modeling of Computer Systems*, pages 265–276, 2008.

[6] P. Samarati and S. D. C. d. Vimercati. Access control: Policies, models, and mechanisms. In *Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design: Tutorial Lectures*, pages 137–196, 2001.

[7] R. Yavatkar, D. Pendarakis, and R. Guerin. A framework for policy-based admission control. RFC Editor, 2000.

[8] B. Lampson, Protection, Proceedings of the 5th Princeton Conference on Information Sciences and Systems, 1971.

[9] Bell, E. D. and La Padula, J. L., Secure computer system: Unified exposition and Multics interpretation, Mitre Corporation, 1976.

[10] David F. Ferraiolo and Ravi Sandhu and Serban Gavrila and D. Richard Kuhn and Ramaswamy Chandramouli, Proposed NIST Standard for Role-Based Access Control, 2001.

[11] Ravi Sandhu and Edward Coyne and Hal Feinstein and Charles Youman, Role-Based Access Control Models, IEEE Computer, 29, 2, 1996, 38–4.

[12] Anas Abou El Kalam and Salem Benferhat and Alexandre Miège and Rania El Baida and Frédéric Cuppens and Claire Saurel and Philippe Balbiani and Yves Deswarte and Gilles Trouessin, Organization based access contro, POLICY, 2003.