Aspect-Oriented State Machines for Resolving Conflicts in XACML Policies

Meryeme Ayache[1(✉)], Mohammed Erradi[1], Bernd Freisleben[2], and Ahmed Khoumsi[3]

[1] ENSIAS, Mohammed V University, Rabat, Morocco
meryemeayache@gmail.com, mohamed.erradi@gmail.com

[2] Department of Mathematics and Computer Science, Philipps-Universität Marburg, Marburg, Germany
freisleb@informatik.uni-marburg.de

[3] Department of Electrical and Computer Engineering, University of Sherbrooke, Sherbrooke, Canada
ahmed.khoumsi@usherbrooke.ca

Abstract. Authorization in collaborative systems is defined by a *global policy* that represents the combination of the collaborators' access policies. However, the enforcement of such a global policy may create conflicting authorization decisions. In this paper, we categorize two types of conflicts that may occur in such policies. Furthermore, to resolve these conflicts and to reach a unique decision for an access request, we present an approach that uses XACML policy combining algorithms and considers the category of the detected conflicts. The approach is implemented using aspect-oriented finite state machines.

1 Introduction

XACML (eXtensible Access Control Markup Language) [7] is one of the access control policy languages that support the combination of multiple sub-policies. This combination may create several conflicting decisions. Therefore, XACML proposes four policy combining algorithms (PCAs) [4] to avoid conflicts between multiple policies, namely: *deny-overrides, permit-overrides, first applicable*, and *only one applicable*. These algorithms take, as input, the authorization decision from each policy matching the request and apply some standard logic to come up with a final decision.

The PCAs are currently chosen in advance by the policy administrator and hence they are static and remain available for all kinds of requests. However, in dynamic environments such as hospitals, there is a need to select the PCAs dynamically depending on the context of the request [5]: emergencies, normal interventions, etc. For emergencies, for example, we usually need to adopt *permit-overrides* in order to grant access to different doctors to save lives. In this paper, we propose a strategy to dynamically choose the adequate PCA based on the type of the detected conflicts and the request's context.

A. El Abbadi and B. Garbinato (Eds.): NETYS 2017, LNCS 10299, pp. 166–171, 2017.
DOI: 10.1007/978-3-319-59647-1_13

The paper is organized as follows. Section 2 presents the conflicts' categorization. In Sect. 3, we present our conflict resolution strategy. Section 4 discusses some aspects of the implementation using aspect-oriented finite state machines. Section 5 concludes the work and outlines future work.

2 Conflict Categorization

A security policy consists of a set of filtering rules, where each rule is specified as a triplet (S, O, a_A), S is a set of subjects (human resources: e.g., doctors), O is a set of objects (physical resources: e.g., patient records), $a \in \{p, d\}$ denotes a permission (if $a = p$) or prohibition (if $a = d$, for deny), and A contains the permitted/prohibited actions among read (r), write (w), create (c) and delete (del). Consider, for example, the rule (S, O, p_r), where S represents the generalist doctor of a hospital H_1, and O represents the medical record of a given patient of another hospital H_2. This rule means that S is permitted to read O.

The composition of many security policies may produce a set of conflicts. A conflict occurs when two policies with different decisions are applicable to the same request. We identify two types of conflicts: conflict of modalities, and conflict of fraction permissions [2]. The first type occurs when two different rules assign contradictory authorizations to the same subject to perform an action over a given object. For example, a conflict of modalities occurs between the following two rules: "The generalist doctor of hospital H_1 is permitted to read the medical records (MR) of the patient x in hospital H_2" and "The generalist doctor of hospital H_1 is forbidden to read the medical record (MR) of all the patients of hospital H_2". The second type represents an ambiguity to make a decision. This ambiguity occurs if two rules with different permitted actions (e.g., read and write) match the same request. For instance, "The radiologists can read the electroencephalogram (EEG) of all the patients" and at the same time "The radiologists can write into the EEG of all the patients". In this case, we have a conflict of fraction permissions, because we do not know which policy to apply since the *write* permission overlaps with the *read* permission. The overlap means if the *write* permission is granted, then obviously the *read* permission is granted, too.

3 Conflict Resolution Strategy

In distributed environments, each organization has its own Policy Decision Point (PDP) that decides which permission is granted to a given subject to perform a specific action on a given object. In collaborative systems, a *master PDP* combining the collaborative policies is used. Our proposed conflict resolution strategy is associated to the *master PDP*. The approach consists of three main steps: (a) select the match policies (policies that match the request), (b) combine the policies into one global policy, and (c) detect and resolve the conflicts. If a conflict of fraction permissions is detected, we resolve it by a prioritization of permissions approach (see Sect. 3.1). If a conflict of modality is detected, we

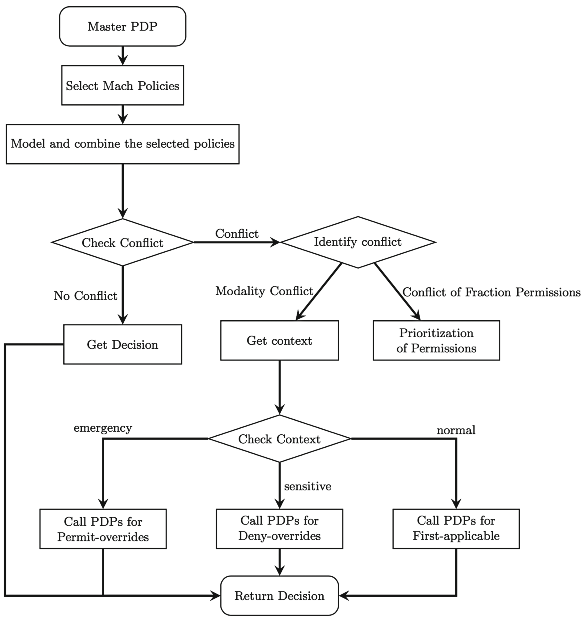
Master PDP
Select Mach Policies
Model and combine the selected policies
Check Conflict
Conflict
Identify conflict
No Conflict
Modality Conflict
Conflict of Fraction Permissions
Get Decision
Get context
Prioritization of Permissions
emergency
Check Context
normal
sensitive
Call PDPs for Permit-overrides
Call PDPs for Deny-overrides
Call PDPs for First-applicable
Return Decision

Fig. 1. Workflow representing the conflict resolution strategy

resolve it by a context-based approach (see Sect. 3.2). A workflow representing the conflict resolution strategy is presented in Fig. 1.

3.1 Prioritization of Permissions Approach

In this approach, the access permissions have distinct priorities. Therefore, for each policy decision associated to several access permissions, we select the one with the highest priority. Table 1 presents two ways of prioritizing permissions, where $>$ denotes *has higher priority than.* The most secure approach is restrictive prioritization. For instance, to resolve the conflict detected in a policy with two different permissions $(p_{r,w}; p_r)$ using restrictive prioritization, we eliminate $p_{r,w}$ and keep p_r. If, on the other hand, we use the permissive conflict prioritization, we eliminate p_r and keep $p_{r,w}$.

Table 1. Prioritization of permissions.

Restrictive prioritization	$p_\emptyset > p_r > p_{r,w} > p_{r,w,c,del}$
Permissive prioritization	$p_{r,w,c,del} > p_{r,w} > p_r > p_\emptyset$

3.2 Context-Based Approach

In the case of a *conflict of modality*, we need to check the request's context. According to the most frequent cases of healthcare, we propose three types of context: "*emergency*", "*sensitive*" and "*normal*". The first context corresponds to the case of emergencies where the patient needs a quick intervention of the doctors to save his life. The *sensitive* context corresponds to patients with sensitive political positions who need to keep their health state secret and there are no emergencies. The *normal* context corresponds to general health cases.

In the case of *emergency*, the *master PDP* chooses *Permit-Overrides* as the appropriate PCA: if one of the policies returns the permit decision, then the *master PDP* permits the access to the required object. If the context is *sensitive*, the chosen PCA is *Deny-Overrides*. Finally, if the context is evaluated to *normal*, the PCA is *First-Applicable*, the master PDP always evaluates the first policy that matches the request.

4 Implementation

To represent security policies, we adopt the automata-based approach that we proposed in our previous work [1]. The approach consists of modeling each security policy by a finite state automaton (or briefly: automaton). We model each rule (S, O, a_A) of a policy by a simple automaton with 3 or 4 states that has two types of transitions: an S-transition is labeled by a set of subjects, and an O-transition is labeled by a set of objects. The authorization a_A is associated with the final state of the automaton. We combine simple automata using the *synchronous product*. The resulting automaton models the security policy.

In this paper, we use aspect-oriented finite state machines (AO-FSM) defined in our previous work [3] to implement our dynamic conflict resolution strategy. An AO-FSM defines a set of states and transition patterns where pointcuts and advices are used to adopt domain-specific language (DSL) [6] state machine artifacts. The pointcuts define matching state (final states) patterns that correspond to the conflicts that may occur in a security policy. For instance, the example of a pointcut in Fig. 2 represents a final state with a fraction permission conflict. As for the advice in Fig. 2, it implements the adequate resolution strategy that consists of removing one of the permissions to avoid the conflict.

Basically, pointcut sub-classes match the current state parameters with the context of a corresponding point of execution in the base code (joinpoint).

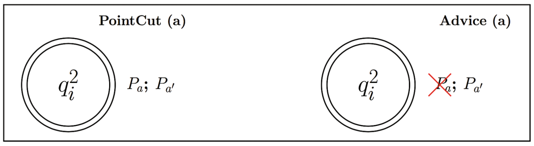

Fig. 2. Examples of aspect artifacts: pointcut and advice

It returns “true” if the pointcut matches, and “false” if not. For instance, the pointcut *FinalState2PermPC*, shown in Listing 1.1, checks if the current state in the pointcut pattern is final. If it is the case, it compares its name with the labels passed in the context. If they are equal, the pointcut matches and returns “true”, otherwise it returns “false”.

The advice language deals with making changes to FSMs to which pointcuts have been matched. The advices implement the resolution strategies of Sect. 3.

Listing 1.1. Excerpt of FinalState2PermPC to define final states with two permissions (pointcut (a) in Fig. 2)

```
public class FinalState2PermPC extends Pointcut {
   ...
      public FinalState2PermPC(String Label1, String Label2) {
     super("pFinalState");
        ...
   @Override
   public boolean match(JoinPoint jp) {
      return ((jp instanceof FinalStateMachineJoinPoint)&&
             (((FinalStateMachineJoinPoint) jp).getFinalStateNames().contains(Label1) &&
             ((FinalStateMachineJoinPoint) jp).getFinalStateNames().contains(Label2)));}}
```

5 Conclusion

In this paper, we have presented two categories of conflicts: fraction of permissions and conflict of modality. We have also presented a conflict resolution strategy that consists of two different approaches: prioritization of permissions and a context-based approach. The selection of the appropriate strategy depends on the type of the detected conflict. The approach uses aspect-oriented finite state machines to intercept, prevent, and dynamically manipulate rules that cause conflicts.

As future work, we intend to integrate the proposed resolution strategy in a cloud environment to evaluate its performance in detecting and resolving conflicts within a large set of policies.

References

1. Ayache, M., Erradi, M., Khoumsi, A., Freisleben, B.: Analysis and verification of XACML policies in a medical cloud environment. Scalable Comput. Pract. Experience **17**(3), 189–206 (2016)
2. Boyland, J.: Checking interference with fractional permissions. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 55–72. Springer, Heidelberg (2003). doi:10.1007/3-540-44898-5_4
3. Dinkelaker, T., Erradi, M., Ayache, M.: Using aspect-oriented state machines for detecting and resolving feature interactions. Comput. Sci. Inf. Syst. **9**(3), 1045–1074 (2012)
4. Lorch, M., Proctor, S., Lepro, R., Kafura, D., Shah, S.: First experiences using XACML for access control in distributed systems. In: Proceedings of the 2003 ACM Workshop on XML Security, pp. 25–37. ACM (2003)

5. Matteucci, I., Mori, P., Petrocchi, M.: Prioritized execution of privacy policies. In: Pietro, R., Herranz, J., Damiani, E., State, R. (eds.) DPM/SETOP 2012. LNCS, vol. 7731, pp. 133–145. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35890-6_10
6. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. (CSUR) **37**(4), 316–344 (2005)
7. Moses, T., et al.: Extensible access control markup language XACML version 2.0. Oasis Standard (2005)