Towards a Full Support of Obligations In XACML

CONFERENCE PAPER · AUGUST 2014

DOI: 10.13140/2.1.2888.0641

5 AUTHORS, INCLUDING:

Donia El Kateb

University of Luxembourg

15 PUBLICATIONS **14** CITATIONS

SEE PROFILE

Yehia Elrakaiby

Fraunhofer Institute for Experimental Softwa…

17 PUBLICATIONS **58** CITATIONS

SEE PROFILE

Tejeddine Mouelhi

University of Luxembourg

33 PUBLICATIONS **270** CITATIONS

SEE PROFILE

Yves Le Traon

University of Luxembourg

236 PUBLICATIONS **2,739** CITATIONS

SEE PROFILE

Towards a Full Support of Obligations In XACML

Donia El Kateb[1,2], Yehia ElRakaiby, Tejeddine Mouelhi[1]
Iram Rubab[1,2] and Yves Le Traon[1,2]

[1]Security, Reliability and Trust, Interdisciplinary Research Center, SnT
[2]Laboratory of Advanced Software SYstems (LASSY)
University of Luxembourg
Luxembourg
donia.elkateb, tejeddine.mouelhi, iram.rubab, yves.letraon@uni.lu
yehia.elrakaiby@gmail.com

Abstract. Policy-based systems rely on the separation of concerns, by implementing independently a software system and its associated security policy.
XACML (eXtensible Access Control Markup Language) proposes a conceptual architecture and a policy language to reflect this ideal design of policy-based systems. However, while rights are well-captured by authorizations, duties, also called obligations, are not well managed by XACML architecture. The current version of XACML lacks (1) well-defined syntax to express obligations and (2) an unified model to handle decision making w.r.t. obligation states and the history of obligations fulfillment/violation. In this work, we propose an extension of XACML reference model that integrates obligation states in the decision making process. We have extended XACML language and architecture for a better obligations support and have shown how obligations are managed in our proposed extended XACML architecture: *OB-XACML*.

Keywords: Usage Control, PEP, PDP, XACML.

1 Introduction

Access control policies regulate users access to the sensitive resources in a given system and they are commonly defined as a set of rules, specified according to an access control model. Obligations [1] allow to extend the notion of access rights with related duties. A complete security policy should encompass both rights and duties, both access authorizations and obligations. XACML (eXtensible Access Control Markup Language) [1] is a standardized policy specification language that defines access control policies in an XML format and defines a standard way to exchange access control requests/responses. Even though XACML supports several profiles to handle authorizations scenarios, its support for usage control is still in its infancy. Indeed, while access control is about a simple boolean decision-making (is the user authorized to access a service/resource?), obligations imply the notion of state. To the best of our knowledge, access control based on obligation states history is not yet handled by XACML architecture.

[1] http://www.oasis-open.org/committees/xacml/

To meet the challenges of reinforcing XACML standard to support obligations, going along the same line than Bertino [2], who pioneered the extension of XACML for usage control, we propose 1) Well-defined XML constructs that are compliant with XACML 3.0 to specify obligations. 2) *OB-XACML*: An underlying architecture that extends the current XACML architecture and that is able to take into consideration the history of obligations fulfillment/violation and obligation states at the level of the decision making process. *OB-XACML* introduces an interaction schema between the different key entities in XACML architecture to keep track of obligations fulfillment/violation related to the users in the system. To the best of our knowledge, our work is a first initiative that considers obligation states as a key element that must be considered at the evaluation time. This paper is organized as follows. Section 2 introduces obligations in XACML. Section 3 describes our extended architecture *OB-XACML*. Section 4 introduces the obligation syntax that supports OB-XACML and describes the different interactions between *OB-XACML* components. Section 5 presents the related work. Finally, Section 6 presents our conclusion and future work.

2 Obligations in XACML

XACML defines obligations as actions that have to be returned to the PEP with the PDP response [2]. XACML defines three PEP categories based on PDP decision and the ability of the PEP to handle obligations.

```
<Obligation ObligationId=''send-email'' FulfillOn=''Deny''>
<AttributeAssignment AttributeId=''email''>donia.kateb@uni.lu</AttributeAssignment>
</Obligation>
```

Listing 1.1. Obligation Example

In the reminder of this paper, we will only consider the *Deny-biased PEP*. In a *Deny-biased PEP* setting, the PEP decision is permit if the PDP decision is permit and all the obligations that are returned by the PDP are fulfilled. In all other cases, the PEP decision is deny. The reader may refer to [2] for more details about other PEPs categories. XACML 2.0 [3] defines obligations as simple attributes assignment that are attached to the policy set or to the policy. In XACML 3.0 [4], obligations can also be added to the rules additionally to the policies and policy sets. An obligation element contains the *obligation identifier* and the *FulfillOn* element which specifies the effect on which the obligation should be fulfilled by the PEP. For example, *FulfillOn* "Permit" specifies that the obligation should be enforced if the PEP decides to permit the request. The XML snippet in Listing 1.1 shows an example that illustrates that if the PDP decision is deny, the subject has to send an email to the address "donia.elkateb@uni.lu".

3 *OB-XACML* Architecture

In an XACML-based architecture, access control decisions are thus taken without taking into consideration obligations fulfillment or violation in previous accesses. In this

[2] http://www.oasis-open.org/committees/xacml/

[3] http://docs.oasis-open.org/xacml/2.0/access control-xacml-2.0-core-spec-os.pdf

[4] http://www.oasis-open.org/committees/xacml/

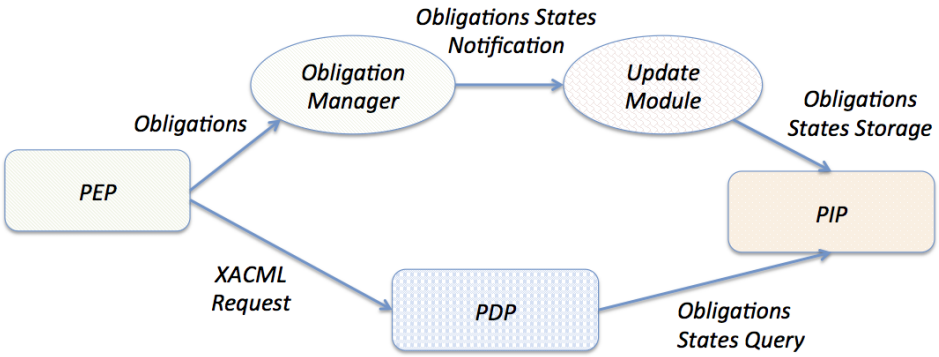

Fig. 1. OB-XACML Workflow

work, we propose to introduce obligation information violations/fulfillment at the decision making time so that access control decisions are taken based on information related to obligation states or related to previous users obligations fulfillment/violation. Here are some motivating scenarios:

- A nurse has to send a report to the patient's treating doctor after each access to the patient's medical data. The report should describe some specific indicators about patient's health status. If the post-obligation of sending a report after the access is violated then the nurse should be prohibited from accessing patient's data in another access and some penalties measures have to be taken against her as a reaction to this non professional behavior.
- A user has the pre-obligation to sign a form before he accesses a web application. If the system keeps track of his fulfilled obligations then the user does not have to sign the form in every session. The system can thus record the fulfillment of the obligation in a first login and then the user can access the system in future sessions without the need to fulfill his pre-obligations.

To support such scenarios, we extend the current XACML architecture so that information inherent from subjects fulfillment/violation of their obligations is taken into consideration at the decision making time. To provide a decision making process that takes into consideration obligation states, we propose to store obligations states in the Policy Information Point (PIP) additionally to attributes values such as resource, subject, environment. Such information is retrieved dynamically at the decision making time and used for request evaluation. Our extension is shown in Figure 1. The PDP sends to the PEP the access decision and the obligations that have to be monitored by the PEP. Each obligation is handled by the obligation manager which tracks obligation states evolution by monitoring their execution in the system. An obligation life cycle can been modeled as a state machine as illustrated in [3]. An obligation state can be 1) Inactive when the fulfillment of the obligation is not needed, 2) Active when the obligation fulfillment is required, 3) Fulfilled when the obligation is satisfied, 4) Violated

when the obligation is violated. 5) Fulfilled/violated when the obligation is violated and later it has been fulfilled 6) An obligation is inactive when it ends. The transition between the different states is driven by contexts [4]. An activation context specifies the different conditions under which the obligation has to be fulfilled whereas the violation context specifies the conditions under which the obligation becomes violated. For example the obligation to send a report after an access to an administration system is handled as follows by the obligation manager: 1) The user has an access to the platform: The obligation is in an *active* state. 2) The user sends the report to the platform: The obligation is in a *fulfilled* state. 3) The user does not send the report within a given time: The obligation is in a *violated* state. The user behavior at the system level is monitored through aspects which capture the different users actions [5]. The update module receives information related to the changes in obligations states and updates the PIP with obligation state attributes that are provided by the obligation manager. We enrich

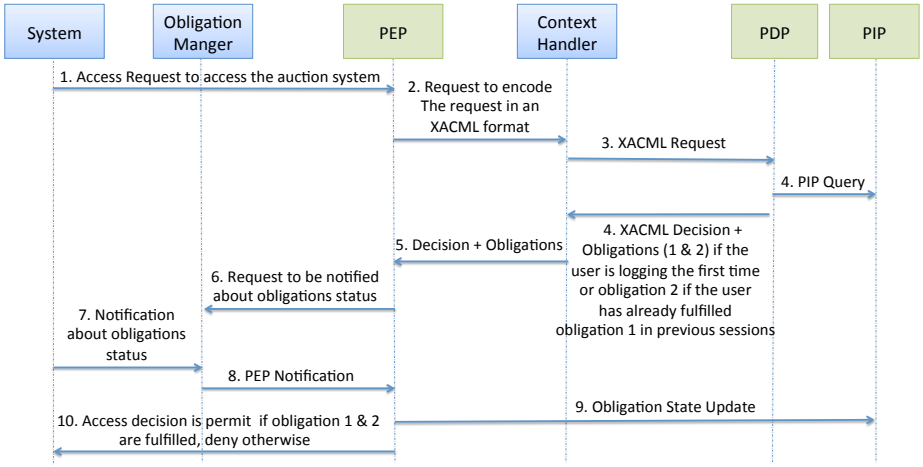

Fig. 2. Pre-Obligations Sequence Diagram

XACML conditions to express access control rules that are conditioned by obligations fulfillment/violation. These conditions will be introduced in the next Section. To explain the processing of obligations in *OB-XACML*, we took some illustrative examples from an Auction Management System (AMS). AMS allows users to perform some bidding operations online if they have enough money in their account before starting bidding operations. We consider the following pre-obligations in the AMS system: 1) The user has to accept the usage terms of the auction before joining the auction system. 2) The user has to validate his payment for the session. Pre-obligation 1 has just to be fulfilled in the first login whereas the pre-obligation 2 has to be fulfilled in every session. Figure 2 illustrates message exchange to process the two pre-obligations in AMS. For a given access request, the PEP decision is taken under the assumption that our PEP is a deny-based PEP. Obligations O include pre-obligations o_{pre}, post-obligations o_{post}

and/or ongoing obligations $o_{ongoing}$. The algorithm 1 specifies how access control decision are handled by the PEP when the PEP receives an access decision with obligations.

Algorithm 1 - *Obligation Management*

Input: PDP Decision, a set of obligations O, **Output:** PEP Decision
/*PDP decision is provided after all preobligations become in an inactive state*/
for all Preobligations $o_{pre} \in O$ **do**
  o_{pre}.state=active
end for
while o_{pre}.state $\neq$ inactive $\forall\ o \in O$ **do**
  /*The PEP is deny-based, all preobligations need to be fulfilled to permit the access if PDP decision is permit */
  if o_{pre}.state=violated **then**
    return Deny
  end if
  PIP update with (o_{pre}.id, o_{pre}.state, o_{pre}.subject, o_{pre}.update_time)
end while
if PDP Decision $\neq$ Deny **then**
  return Permit
else
  return Deny
end if
/*PDP decision is revoked if some ongoing obligations are violated*/
for all Ongoing obligations $o_{ongoing} \in O$ **do**
  $o_{ongoing}$.state=active
end for
while $o_{ongoing}$.state $\neq$ inactive $\forall\ o_{ongoing} \in O$ **do**
  if $o_{ongoing}$.state=violated **then**
    PDP Decision = Deny
  end if
  PIP update with ($o_{ongoing}$.id, $o_{ongoing}$.state, $o_{ongoing}$.subject, $o_{ongoing}$.update_time)
end while
/*Postobligations do not impact PEP decision*/
for all Postobligations $o_{post} \in O$ **do**
  o_{post}.state=active
end for
while o_{post}.state $\neq$ inactive $\forall\ o_{post} \in O$ **do**
  PIP update with (o_{post}.id, o_{post}.state, o_{post}.subject, o_{post}.update_time)
end while

4 *OB-XACML* Language and Proposed Architecture

This Section introduces the obligation syntax that supports *OB-XACML* and describes the different interactions between *OB-XACML* different components.

4.1 Obligations Syntax in *OB-XACML*

XACML defines obligations as simple attribute assignment. To specify obligations in XML, we have leveraged an existing formal obligation model [3] to identify the dif-

ferent elements of an obligation. In XACML 3.0, users are able to extend XACML syntax and to define their own categories, we added new identifiers to define obligation elements in XACML 3.0. To refer to the entity that is responsible of enforcing an obligation, we introduce a new Attribute identifier: the obligation-subject encoded as shown in Listing 1.2:

```
<Category=urn: oasis: names: tc: xacml:
    1.0:subject-category:obligation-subject>
```

Listing 1.2. Obligation Subject

Similarly, action and resource identifiers are added using *action-category:obligation-action* and *resource-category:obligation-object*. We distinguish between the obligations that have to be performed in each session by a given subject and those that have just to be performed by the first login using the identifier "each session" to specify obligations that have to be fulfilled in every access and "first login" to specify obligations that have just to be fulfilled in the first access.

4.2 Description of *OB-XACML* Components

Figure 3 illustrates the interactions between the different components in *OB-XACML*:

a) AMS application: We consider an Auction Management System (AMS) which is a Java policy-based application which contains 122 classes and 797 methods.

b) Obligation Manager: The obligation manager receives obligations from the PEP and maintains their states. It includes two mapping modules:

- A Mapper from abstract obligations to concrete obligations: This modules translates obligation parameters included in XACML obligations to parameters that are interpreted at the application level. For instance a required action in an obligation is translated to a method call that triggers some functionality at the business level logic, a role is mapped to a user, etc.
- An Obligation States Monitor: For obligations state monitoring, we define abstract rules that describe the impact of application parameters on obligation states. For example, the obligation to put a starting bid before joining an auction session evolves from an active state to a fulfilled one when the user validates the payment. This requirement is described by the rule $\Re_i$ that describes the operations needed for the obligation "joining an auction" to transition from an active state to a fulfilled state:

$\Re_i$: State(Obl_1:joining an auction, active) $\mapsto$ State(Obl_1:joining an auction, fulfilled)
If call method(Validate Bid.amount()) && Bid.amount(subject s) returns amount && amount $>$ allowed_minimum_seuil)

To monitor the different parameters related to obligations state changes which are defined in our mapping rules, we use aspect oriented programming [5]. The obligation manager is a Java module that monitors a set of events. Each obligation is a Java class that extends an abstract class event.

c) PIP attributes database: We implemented the PIP using a MySQL database that is updated with records describing obligation parameters whenever a change in an obliga-

tion state is reported using following form: (Obligation_ID, Obligation_Subject, Obligation_Object, Obligation_Action, Time, Obligation State). This database is queried by the PDP during access requests to fetch information related to the obligations status or related to obligations violation/fulfillment.

d) Update module component: The update module is triggered by aspects in each obligation state change and it updates the PIP with obligations state attributes.

e) Extended PDP: To extend Sun's XACML implementation with a PDP that supports the new types and the new attributes that we have defined in this work, we have extended XACML standard factory [5] with our new factory.

f) Timer: We use a Java timer to implement a timer that starts when the activation context starts, the violation context starts when the timer expires and the obligation is not fulfilled.

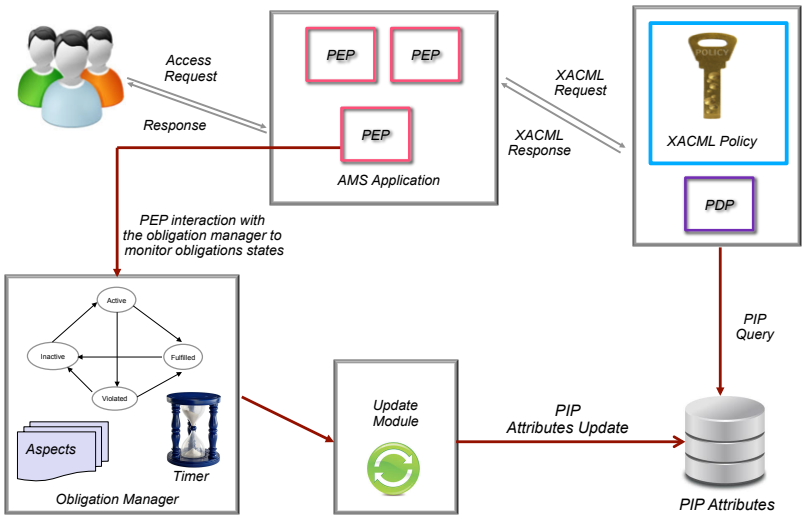

Fig. 3. *OB-XACML* Workflow

5 Related Work

In the last few years, several research initiatives have motivated the support of obligations in XACML at the level of XACML language and architecture. In [6], the authors have proposed a framework and a supporting language extending XACML to take into consideration UCON features. They have thus added some identifiers to XACML reference language to support mutability of attributes and identifiers to handle the different access phases, thus taking into consideration the continuity of access features in UCON model. The work presented in [7] [8] follows the same direction and aims at enriching XACML model to take into consideration UCON features by adding the identifier in the

[5] http://sunxacml.sourceforge.net/guide.html

condition element to distinguish between pre-obligations, ongoing and post-obligations. The element *AttrUpdates* is added to reason about attributes update and an XML retrieval policy has been introduced to specify where the attributes have to be retrieved for update. In [2], the authors have proposed a language and an underlying architecture to handle obligations. Obligations are commonly defined as application-specific and thus their handling is left to the platform that manages them. Their proposed obligation schema includes a list of event families that categorize events types interacting with an obligation. To the best of our knowledge, our XACML extend model is a first initiative that considers the user's history of obligation violation/fulfillment information at the decision making time.

6 Conclusion

The work that we are in presenting in this paper goes in a research direction that we are currently investigating, which spans over policy-based software architectures and particularly those that are based on XACML [9,10]. In this paper, we propose a syntax to support obligation polices in XACML and an extension of the standard XACML architecture to take into consideration obligations states and information related to their violation/fulfillment in the decision making process. The changes that we have introduced at the level of XACML architecture do not require to perform many modifications at the level of XACML reference model, which eases the adoption of *OB-XACML*. Our perspectives to extend this work are twofold: (1) At the level of XACML policy language, we plan to extend the language and to define a priority order between obligations so that the PEP is able to handle obligations returned by the PDP according to the obligations priority strategy stated in the policy. (2) At the level of *OB-XACML* model, we need to analyze the impact of messages exchange between the PEP and the PIP on the overall performance of real-life policy-based systems.

References

1. X. Zhang, "Formal model and analysis of usage control," Ph.D. dissertation, 2006.
2. N. Li, H. Chen, and E. Bertino, "On practical specification and enforcement of obligations," in *CODASPY*, 2012, pp. 71–82.
3. Y. Elrakaiby, F. Cuppens, and N. Cuppens-Boulahia, "Formal enforcement and management of obligation policies," *Data Knowl. Eng.*, 2012.
4. Y. Elrakaiby, T. Mouelhi, and Y. L. Traon, "Testing obligation policy enforcement using mutation analysis," in *ICST*, 2012, pp. 673–680.
5. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP*, 1997, pp. 220–242.
6. U. e Ghazia, R. Masood, M. A. Shibli, and M. Bilal, "Usage control model specification in xacml policy language," in *CISIM*, 2012, pp. 68–79.
7. F. M. Maurizio Colombo, Aliaksandr Lazouski and P. Mori, "A proposal on enhancing xacml with continuous usage control features," in *Grids, P2P and Services Computing*, 2010.
8. A. Lazouski, F. Martinelli, and P. Mori, "A prototype for enforcing usage control policies based on xacml," in *TrustBus*, 2012, pp. 79–92.
9. D. E. Kateb, T. Mouelhi, Y. L. Traon, J. Hwang, and T. Xie, "Refactoring access control policies for performance improvement," in *ICPE*, 2012, pp. 323–334.
10. J. Hwang, T. Xie, D. E. Kateb, T. Mouelhi, and Y. L. Traon, "Selection of regression system tests for security policy evolution," in *ASE*, 2012, pp. 266–269.