Towards an Efficient Policy Evaluation Process in Multi-Tenancy Cloud Environments

Meryeme Ayache Mohammed Erradi Bernd Freisleben* Ahmed Khoumsi#

ENSIAS, Mohammed V University in Rabat, Morocco
* Dep. of Math. & Comp. Sc., Philipps-Universität Marburg, Germany
# Dep. of Elec. & Comp. Eng., University of Sherbrooke, Canada
{meryemeayache,mohamed.erradi}@gmail.com,
freisleb@informatik.uni-marburg.de,
ahmed.khoumsi@usherbrooke.ca

ABSTRACT

Cloud computing offers most of its services under multi-tenancy environments. To satisfy security requirements among collaborating tenants, each tenant may define a set of access control policies to secure access to shared data. Several cloud solutions make use of XACML to specify such policies. However, existing implementations of XACML perform a brute force search to compare a request to all existing rules in a given XACML policy. This decreases the decision process (i.e., policy evaluation) performance especially for policies with a large number of rules. In this paper, we propose an automata-based approach for an efficient XACML policy evaluation. We implemented our approach in a cloud policy engine called X2Automata. The engine first converts both XACML policies and access requests to automata. Second, it combines the two automata by a synchronous product. Third, it applies an evaluation procedure to the resulting automaton to decide whether an access request is granted or not. To highlight the efficiency of X2Automata, we compare its performance, based on the OpenStack cloud environment, with the XACML implementation named *Balana*.

Keywords

Multi-Tenancy; Cloud Computing; Access Control; XACML; Automata.

1. INTRODUCTION

To facilitate cross-organizational collaborations, cloud computing offers multi-tenancy architectures. In such architectures, multiple customers, called tenants, transparently share the cloud's resources and services [6]. Multi-tenancy permits cloud providers to establish collaborative relationships among cloud users [1], especially at the storage service level. Thereby, the data stored in the cloud are available not only to its original owners, but also to users of other tenants. However, data across collaborations raises significant challenges in terms of security and especially access control.

To satisfy security requirements, each collaborating tenant has to define a set of security policies. For this purpose, most cloud environments make use of XACML (eXtensible Access Control Markup Language) [8] to specify access control policies. XACML rules consist of subjects (e.g., doctors), resources (e.g., patient's records), an effect (e.g., permit/deny) and an action (e.g., read/write). An XACML request represents a subject's request to perform a specific action on a specific resource. XACML assumes an architecture containing a PDP (Policy Decision Point) that searches in the policy repository for the appropriate policy that matches the request. The PDP then sends a response to the requestor, which can be: *Permit*, *Deny*, *Not Applicable* or *Indeterminate*. *Not Applicable* is applied if no rule matches the request. While *Indeterminate* is applied if the system cannot interpret the request. This process is called *decision process* or *policy evaluation process*. However, commercial implementations of XACML policy evaluation engines such as Sun XACML PDP perform a brute force search to compare an access request to all the rules in an XACML policy. This search technique decreases the policy evaluation performance.

In this paper, we present a novel approach to increase the evaluation performance of XACML. The approach is implemented as a cloud policy engine denoted *X2Automata*. It translates XACML policies and access requests into automata, and combines the two automata by a synchronous product. Then, it applies an evaluation procedure to the resulting automaton to decide whether an access request is granted or not. Thus, instead of checking all XACML policy rules, we check only the final states of the combined automaton. Such states represent the permitted actions. To demonstrate the efficiency of our approach, we evaluate its performance against the XACML open-source implementation named *Balana*. The implementation and the performance measurements of our approach are based on the OpenStack cloud environment.

The remainder of the paper is organized as follows: The related work is summarized in Section 2. In Section 3, we present preliminary remarks on automata. Section 4 presents a collaborative medical scenario used to illustrate the proposed approach. Section 5 is devoted to the automata-based description of XACML requests and policies. The description of the proposed policy evaluation process is presented in Section 6. Section 7 discusses the implementation of X2Automata, a policy engine based on the proposed approach. Section 8 presents experimental results compared to *Balana*, the only XACML open-source implementation. Finally, Section 9 concludes the paper and outlines future work.

2. RELATED WORK

Several approaches have been proposed to improve the policy evaluation time in XACML. For instance, Liu et al. [7] proposed

CCSW'16, October 28 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2996429.2996431

XEngine, a scheme for XACML policy evaluation. It consists of three steps: (1) converting a textual XACML policy to a numerical policy; (2) normalizing the policy by converting the output of the first step to a hierarchical structure and converting the combined algorithm to first-applicable; (3) converting the normalized numerical policy to a tree data structure for efficient processing of requests. The approach improves the performance of the XACML Sun PDP (Policy Decision Point) by three to four orders of magnitude. The pre-processing time of XEngine includes the time for normalizing the policy and the time for building the internal data structure. In our approach, the pre-processing step represents the automata construction for the XACML policy. The pre-processing procedure is optional in our approach and required only during policy deployment. However, the XEngine approach requires this step because it is the core of the proposal to improve the evaluation response time.

Mourad and Jebbaoui [9] proposed SBA-XACML, a scheme that provides the evaluation of XACML policies. SBA-XACML is a set-based language composed of all the elements and constructs needed for the specification of XACML based policies. It is a mathematical intermediate representation of policies based on set theory. SBA-XACML ameliorated the techniques of XEngine, but it remains scalable for evaluating requests in a single independent organization. The experimental results profit from the fact that the SBA-XACML evaluation of large and small size policies offers a better performance than XEngine and Sun PDP, by a factor ranging between 2.4 and 15 times faster depending on the policy size. However, our experiments in Section 7 show that our approach even outperforms the SBA-XACML approach.

Ngo et al. [10] presented mechanisms to transform XACML operations into decision tree diagrams, aimed at optimizing policy evaluation. Their proposed approach is interesting, however, the authors have only experimented on small scale policies with up to 360 rules, unlike our approach that used policies with 10000 rules.

Our automata-based approach consists of three steps: a) translating each access control policy into automata; b) combining the security policies using the synchronous product; the resulting automaton is the global security policy of all the collaborators - if a new security policy is added to the collaboration, the system transforms it into an automaton and combines it with the global automaton of the previous policies; c) translating the requests into automata; the combination of the two automata representing the request and the policy, allows us to check whether the request is permitted or not.

3. PRELIMINARIES

Finite state automata (or briefly automata) can be formally defined by $\mathscr{A} = (\Sigma, Q, q^0, Q^f, \delta)$ where Σ is a finite set of events (also called alphabet), Q is a finite set of states, q^0 is the initial state and $Q^f \subseteq Q$ is a finite set of final states. $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, where $\delta(q,\sigma) = r$ means that the execution of the event σ (or the reading of the term σ) from state q leads to state r. $\delta(q,\sigma) = r$ can also be written as $q \xrightarrow{\sigma} r$.

An automaton $\mathscr{A}$ consists of states linked by labeled transitions, and represented by a graph whose nodes and arcs are the states and the transitions of $\mathscr{A}$, respectively. There is one initial state (with a small incoming arrow) and one or more final states (double circled).

The theory of automata allows us to compose models of systems, behaviors, mechanisms due to the operations that can be performed over automata. For instance, the synchronous product of two automata $\mathscr{A}_1$ and $\mathscr{A}_2$ over the alphabet Σ is an automaton over the alphabet Σ whose language is $\mathscr{L}_{A_1} \cap \mathscr{L}_{A_2}$.

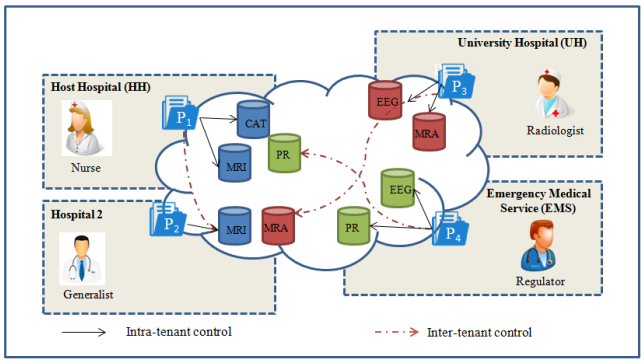

Figure 1: Multi-tenant outsourced data sharing

4. MULTI-TENANCY COLLABORATION

To motivate the problem of controlling outsourced data sharing, we consider a collaborative medical application scenario illustrated in Figure 1. In the example, we consider four medical organizations, namely: host hospital (HH), hospital 2 (H), emergency medical service (EMS), and university hospital (US). The organizations are collaborating with each other in order to perform a remote diagnosis of a patient in the host hospital (the hospital where the patient has been transferred). Doctors located in HH can make use of the experiences of specialists located in other medical organizations. Each organization, i.e., a tenant, manages its internal resources via an access control policy (intra-tenant control). A given medical file may be stored in two different organizations. For instance, a MRI (Magnetic Resonance Imaging) scan is stored in two different tenants: HH and hospital 2. The cross-tenancy communication imposes an inter-tenant management. This kind of control consists of combining the authorization policies in order to control the access of the shared data.

Each organization has: i) a set of subjects: they are human resources, b) set of objects: they are physical and computer resources (hardware, software), and c) a security policy to regulate the accesses to its objects.

An access request can be formally described as the triplet: (S, O, a), where S is a set of subjects, O is a set of objects of a single organization, and a is an action which can be read, write, create or delete. (s, o, a) means that a subject s $\in$ S applies the action a to an object o $\in$ O. For instance, the request of a radiologist from university hospital to read the MRI scans located in hospital 2 can be described by ($radiologist_{\text{EMS}}$, MRI_{H_2}, read). Note that we introduce the organization name as an index.

A security policy consists of a set of filtering rules, where each rule is specified as a triplet (S, O, p_A), S is a set of subjects, O is a set of objects, and p_A denotes a permission or prohibition (permit or deny), and A contains the permitted actions among read, write, create and delete. It is worth noting the similarity between a filtering rule (S, O, p_A) and an access (S, O, a). The semantic difference is that an access indicates the effective application of an action a, while a rule indicates the permission of applying any action of A.

We use the notation $\neq$S to denote the complementary of S with respect to (w.r.t) all the involved organizations, and $\neq$O to denote the complementary of O w.r.t. the organization owning the objects in O. We will also use the notations $doctors_{\neq x}$, $nurses_{\neq x}$ and $radiologists_{\neq x}$ to denote the doctors, nurses and radiologists of all the organizations excluding x, respectively. Note, for example, that $\neq doctors_x$ and $doctors_{\neq x}$ do not have the same meaning; the for-

Table 1: Example of a XACML policy

RuleID	Effect	SubjectMatch	ResourceMatch	ActionMatch
R_1	Permit	generalist	PR	read, write
R_2	Permit	neurologist	EEG	read
R_3	Permit	radiologist	Scans	write
R_4	Deny	radiologist	Scans	write
R_5	Deny	generalist	PR	read, write
R_6	Permit	neurologist	EEG	read
R_7	Permit	generalist	PR	read, write

mer denotes all subjects but the doctors of x, while the latter denotes all the doctors of any organization but x. For a subject s and an object o, we say that (s, o) matches a rule R=(S, O, p_A) (we also say that R matches (s, o)) if s $\in$ S and o $\in$ O.

For the rest of the paper, we consider the security policy of the emergency medical service (EMS) shown in Table 1 as an example.

5. AUTOMATA-BASED DESCRIPTION

In XACML, a rule is described by: an *Effect* and a *Target*. The *Effect* can have two values: "*Permit*" and "*Deny*". The *Target* is a combination of *Match* elements. Each *Match* element describes an attribute that a *Request* should match in order to activate a policy. There are four attribute categories, namely: subject, resource (object), action and environment. Our proposed automata-based approach is realized as follows: From the XACML representation of a policy F, we construct an automaton $\mathscr{A}$ that models F, and then our analyses of F are done on $\mathscr{A}$. The automaton $\mathscr{A}$ generated from a policy F has the following characteristics: from the initial state of $\mathscr{A}$, we have several possible paths where each path consists of *a pair* of transitions that leads to a final state associated with a permission p_A (see Sect. 4). Each path represents a rule (S, O, p_A) of F as follows: the first and second transitions are labeled S and O, respectively, and the reached state is associated with p_A. The construction of the automaton from the policy is performed by a *synthesis procedure* defined in detail in our previous work [5]. It consists of four steps:

- **Step 1**: extraction of attributes from each XACML rule: s, o, p, A. For example, we consider $Rule_1$ of Table 1, s = generalist, o = PR, p = permit, and A = {read, write}.
- **Step 2**: each rule is represented by an automaton with four states x_i^0, x_i^1, and x_i^2 and $x_i^\#$, where x_i^0 represents the initial state, x_i^2 (match state) and $x_i^\#$ (no-match state) are final states. x_i^0 and x_i^1 are linked by a transition labeled S, and the pair of transitions x_i^1 and x_i^2 is linked by a transition labeled O. Transitions labeled $\neq$S and $\neq$O link x_i^0 and x_i^1 to $x_i^\#$, respectively. The permission p_A is associated with the final state x_i^2. For instance, $Rule_1$ of Table 1 can be represented by the automaton in Figure 2.

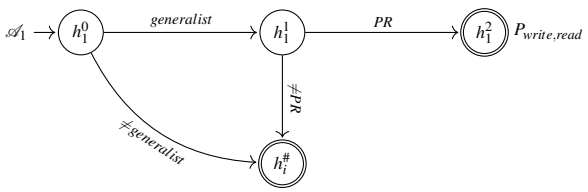

Figure 2: Automaton $\mathscr{A}_1$ obtained in Step 1 for the rule R_1.

- **Step 3**: the automata constructed in step 2 do not have the same alphabet. Therefore, the objective is to rewrite the transitions of the automata so that they have the same alphabet. This is realized by partitioning each of the domains of subjects and objects into a set of disjoint sets. This partitioning permits us to express a transition of an automaton as a union of sets of the partition.
- **Step 4**: In order to model the security policy defined in a XACML policy file, we combine the automata resulting from Step 3 by an operator called *synchronous product*. The resulting automaton representing the policy of an organization x is denoted $\mathscr{A}_x$.

To describe the XACML request by an automaton, we follow steps 1, 2, and 3 of the *synthesis procedure*. A slight difference between the two descriptions is that in step 3 the final state is associated with an action instead of permission. Due to space limitations, we do not present the resulting automaton of the access request.

6. XACML POLICY EVALUATION

The objective of this section is to check whether a request satisfies an XACML policy; this process is called *XACML policy evaluation*. We use the following definitions:

DEFINITION 1. *An access to an organization x is any access=(S, O, a) such that O consists of objects of x.*

DEFINITION 2. *An access* Acc *= (S, O, a) to an organization x* ***respects*** *a security policy $SecPol_x$ if for every s $\in$ S and o $\in$ O, $SecPol_x$ permits that s has access to o through the action a of* Acc.

From the fact that the security policy of an organization x specifies restrictions uniquely on objects of x, our analysis is achieved by verifying whether the security policy $SecPol_x$ of each organization x is respected by every access to x. Our objective can therefore be achieved by the following procedure:

Analysis Procedure
for every organization x:
  for every access Acc of the collaborative session:
    if Acc is an access to x then:
      verify if Acc respects $SecPol_x$

We now propose an automata-based approach to execute the "verify if Acc respects $SecPol_x$" instruction of the analysis procedure. Let $\mathscr{A}$ and $\mathscr{A}_x$ be the automata modeling Acc = (S, O, a) and $SecPol_x$ according to Section 5, respectively. Recall that $\mathscr{A}$ has a match state and possibly a no-match state, and $\mathscr{A}_x$ has one or several match states. We have the following proposition:

PROPOSITION 1. *Consider a security policy $SecPol_x$ of an organization x and an access* Acc *to x. We have Acc respects $SecPol_x$ if for every (s, o):*

- *(s, o) leads to the match state of $\mathscr{A}$ (associated to the action a) implies that*
- *(s, o) leads to a match state of $\mathscr{A}_x$ associated to a permission p that contains a, i.e., a $\in$ p.*

The verification of the implication of Proposition 1 is achieved in two steps that are similar to the Steps 3 and 4 of the synthesis procedure of Section 5.

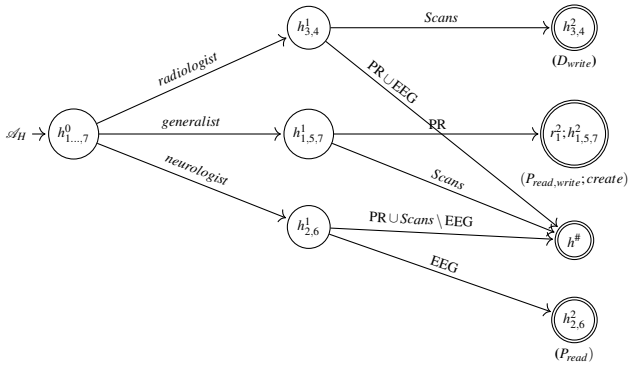

Figure 3: Automata $\mathscr{A}_H$ modeling the policy of Table 1.

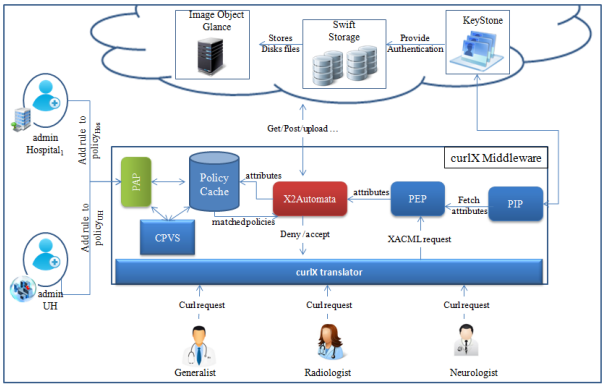

Figure 4: Interaction between different components of the curlX middleware

Step a: The automata $\mathscr{A}$ and $\mathscr{A}_x$ may have distinct alphabets because their respective transitions are not necessarily labeled by the same sets of subjects and objects. To be able to combine these automata in Step b, the objective of Step a is to re-label the transitions of $\mathscr{A}$ and $\mathscr{A}_x$ so that they have the same alphabet. This is achieved by splitting each transition labeled by a given set U (of subjects or objects) into several transitions labeled by an adequate partition of U. The states are not modified.

Step b: After their transformation in Step a, $\mathscr{A}$ and $\mathscr{A}_x$ are then combined by synchronous product. The resulting automaton is denoted by $\mathscr{A} \times \mathscr{A}_x$. Each of its states is a combination (r, q) of two states of $\mathscr{A}$ and $\mathscr{A}_x$, respectively. In particular, (r, q) is a final state of $\mathscr{A} \times \mathscr{A}_x$ if q and r are final states of $\mathscr{A}$ and $\mathscr{A}_x$.

We have the following proposition used as an automata-based procedure to execute "verify if *Acc* respects $SecPol_x$" instruction of the analysis procedure.

PROPOSITION 2. *Consider a security policy $SecPol_x$ of an organization x and an access Acc to x. Let $A \times \mathscr{A}_x$ be the synchronous product of the automata $\mathscr{A}$ and $\mathscr{A}_x$ modeling Acc = (S, O, a) and $SecPol_x$, resp. We have Acc respects $SecPol_x$ if for every final state (r, q) of $\mathscr{A} \times \mathscr{A}_x$:*

- *r is a match state of $\mathscr{A}$ (hence r is associated to the action a) implies that*
- *q is a match state of $\mathscr{A}_x$ associated to a permission p containing the action a, i.e. $a \in p$.*

As an example, let us consider that a generalist wants to create a personal record (PR). We apply the automata-based procedure to "verify if *Acc* respects $SecPol_H$". We obtain the automaton $\mathscr{A} \times \mathscr{A}_H$ (Figure 3) which has a match state named $(r_1^2; h_{1,5,7}^2)$ that corresponds to $(create; p_{r,w})$. Using Proposition 2, we deduce that *Acc* does not respect $SecPol_{H_1}$ because "create" $\notin p_{r,w}$.

7. IMPLEMENTATION

In this section, we discuss implementation issues of the proposed approach in OpenStack (an open-source cloud platform). OpenStack provides several services, namely: Computation (*nova*), identity (*keystone*), object storage (*swift*), block storage (*cinder*), image (*glance*) and dashboard (*horizon*). OpenStack regroups its users by domains, projects or groups. We assume that each domain represents a medical organization in OpenStack while projects inside a domain could represent a diagnosis in the medical organization. In the case of collaborations, multiple organizations would form a group. Information about each group including ID, users, domains, etc. are stored in keystone.

We implemented our automata-based approach in the OpenStack Devstack release. We integrated, in a previous work, a middlware denoted curlX [3] to manage access control policies in Openstack. In this paper, we replaced the XACML PDP of curlX, by the new policy evaluation engine X2Automata shown in Figure 4. curlX is composed of three main components: a) *curlX translator* [4]: responsible for translating the cloud client's request into XACML request; b) *CPVS* [5]: detects conflicts and redundancies in the XACML policies; note, for example, that the policy shown in Table 1 contains many conflicts and redundancies, and CVPS is responsible for detecting them and then a resolution strategy is applied to eliminate them; c) *X2Automata*: is responsible of evaluating the XACML request using the automata-based approach presented in Section 6; it translates the policies into automata and combines them to have a global security policy in the case of collaborative applications. X2Automata is also responsible of translating the XACML requests into automata for the evaluation process.

8. EXPERIMENTAL EVALUATION

Our experiments were performed on an Intel Core 2 Duo CPU 2 GHz with 3 RAM running on Windows 7. It is difficult to get a large number of real-life XACML policies, since access control policies are often deemed confidential for security reasons. Therefore, we have developed a random XACML policy generator. We generated 6 synthetic XACML policies of large sizes. In our experiments, we evaluated the impact of the policy size in terms of the number of rules.

Fig. 5 shows the pre-processing time versus the number of rules for X2Automata. The pre-processing time represents the time for constructing the policy's automaton. We observe that there is an almost constant correlation between the number of rules and the pre-processing time of X2Automata for policies with hundreds of rules, which demonstrates that X2Automata is scalable in the pre-processing phase for policies with a small number of rules. However, it starts to grow for large size policies due to the multiplication of the number of rules. This increase is also related to the number of subjects and objects in each security policy. However, this pre-processing step is executed only if the policies are modified: new rules are added, new policies are added to the collaboration, or rules are changed in a given policy. In the first case, the time of pre-processing almost does not change because we combine the rule's

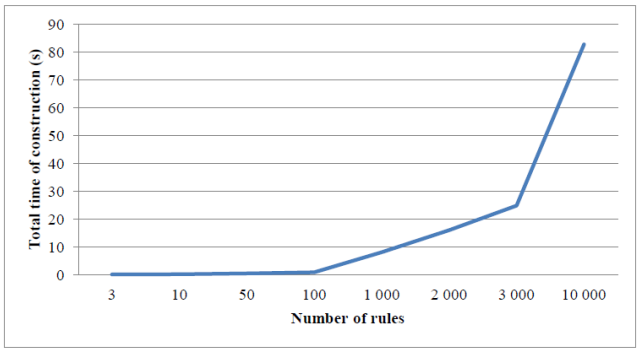

Figure 5: Pre-processing (Automata construction) time on synthetic XACML policies.

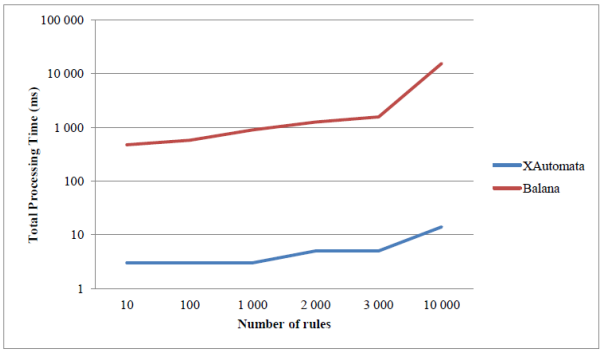

Figure 6: Processing time difference between Sun PDP and X2Automata.

automaton with the existing policy's automaton. However, the two other cases require the entire pre-processing time. For instance, let us consider an organization x with a policy $PolSec_x$ containing 1000 rules. This organization is collaborating with another organization denoted y with a policy $PolSec_y$ containing 2000 rules. The time to add the new policy $PolSec_y$ to $PolSec_x$ is around 9 seconds. This time includes the pre-processing time to generate the automaton of $PolSec_y$ and the time of the synchronous product. In fact, the time of 9 seconds is dominated by the generation of the automaton of $PolSec_y$, because the time for the synchronous product does not exceed 5 ms.

In terms of efficiency, we measured the request processing time of X2Automata compared to Balana[1]. For X2Automata, the processing time for a request includes the time for constructing the synchronous product of the request's automaton and the policy, and the time for finding the final state that matches the request. For Sun PDP (Balana), the processing time for a request is the time for finding the decision in the XML file. Fig. 6 shows the difference between Sun PDP and X2Automata for the total processing time of different synthesized XACML policies. Note that the two lines in Fig. 6 are not close to each other. This figure shows that X2Automata outperforms Sun PDP by 10 times.

9. CONCLUSION

This paper has presented an automata-based approach to improve the XACML policy evaluation engine. The approach has been implemented in a cloud environment as a policy evaluation engine called *X2Automata*. The approach consists in describing both an XACML policies and access requests by automata. Their synchronous product facilitates the policy evaluation process. We have performed a comparison with Sun PDP implemented by Balana. The conducted experiments have shown that X2Automata policy evaluation of large and small size policies has better performance than Sun PDP.

As future work, we intend to extend our automata-based approach to support other rule-based policy languages such as Enterprise Privacy Authorization Language (EPAL)[2].

Acknowledgments

This work is supported by the BMBF (PMARS Program) and the DAAD (German-Arab Transformation Partnership).

[1]The only open-source implementations of XACML 3.0: http://xacmlinfo.com/category/balana/.

References

[1] M. Almorsy, J. Grundy, and A. S. Ibrahim. Collaboration-based cloud computing security management framework. In *2011 IEEE International Conference on Cloud Computing*, pages 364–371. IEEE, 2011.

[2] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal). *IBM Re-search*, 2003.

[3] M. Ayache, M. Erradi, and B. Freisleben. Access control policies enforcement in a cloud environment: OpenStack. In *2015 11th International Conference on Information Assurance and Security (IAS)*, pages 26–31. IEEE, 2015.

[4] M. Ayache, M. Erradi, and B. Freisleben. curlx:Amiddleware to enforce access control policies within a cloud environment. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 771–772. IEEE, 2015.

[5] M. Ayache, M. Erradi, A. Khoumsi, and B. Freisleben. Analysis and verification of XACML policies in a medical cloud environment. *Scalable Computing: Practice and Experience*, 17(3):189–206, 2016.

[6] J. Kabbedijk, C.-P. Bezemer, S. Jansen, and A. Zaidman. Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective. *Journal of Systems and Software*, 100:139–148, 2015.

[7] A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: a fast and scalable xacml policy evaluation engine. *ACM SIGMETRICS Performance Evaluation Review*, 36(1):265–276, 2008.

[8] T. Moses et al. Extensible access control markup language (xacml) version 2.0. *Oasis Standard*, 200502, 2005.

[9] A. Mourad and H. Jebbaoui. SBA-XACML: set-based approach providing efficient policy decision process for accessing web services. *Expert Systems with Applications*, 42(1): 165–178, 2015.

[10] C. Ngo, Y. Demchenko, and C. de Laat. Decision diagrams for XACML policy evaluation and management. *Computers & Security*, 49:1–16, 2015.