Modeling Authorization Policies for Web Services in Presence of Transitive Dependencies

Worachet Uttha[1], Clara Bertolissi[1,2] and Silvio Ranise[2]
[1]*LIF, CNRS UMR 7279 & AMU, Marseille, France*
[2]*Fondazione Bruno Kessler, Trento, Italy*

Keywords: Access Control, Transitive Access, Security Policy, OrBAC, Web Services, XACML.

Abstract: Access control is a crucial issue for the security of Web Services. Since these are independently designed, implemented, and managed, each with its own access control policy, it is challenging to mediate the access to the information they share. In this context, a particularly difficult case occurs when a service invokes another service to satisfy an initial request, leading to indirect authorization errors. To overcome this problem, we propose a new approach based on a version of ORganization Based Access Control (OrBAC) extended by a delegation graph to keep track of transitive authorization dependencies. We show that Datalog can be used as the specification language of our model. As a byproduct of this, an automated analysis technique for simulating execution scenarios before deployment is proposed. Finally, we show how to implement an enforcement mechanism for our model on top of the XACML architecture. To validate our approach, we present a case study adapted from the literature.

1 INTRODUCTION

In the Service Oriented Architecture (SOA) paradigm, applications can be given a standard interface and stored as reusable units for composition. This loosely-coupled architecture makes platform-independent computing possible but subsumes all the problems and issues of distributed computing (such as non-determinism and synchronization) while adding, among others, a number of additional problems related to the dependability of the services. Dependability refers to the property of a system by which some trust can be placed on the delivered service. Since SOA systems are usually obtained by composition of simpler services, dependability is the result of the capability of the component services to deliver certain results and the way these results are exchanged among the services. This aspect is particularly important when security properties come into the picture and authentication or authorization of users with respect to the integrated services must be enforced. While for authentication, there exist standardized and thoroughly analyzed solutions—such as the Single-Sign On (Armando et al., 2012) —for authorization, the situation is much less satisfactory. One of the main problems is due to the presence of transitive dependencies, which can be explained as follows. In the processing of a request, a service may call other services to complete its computations. Even though the user had the rights to access the original service, he/she might not have the permissions to access the invoked services, leading to indirect authorization errors. In general, these errors are difficult to prevent through testing once the integrated services have been deployed, because of the large number of possible execution scenarios.

The **first contribution** of this paper is a framework for the modeling of access control policies—at design time—of Web Services, an important class of SOA applications. For concreteness, we develop our ideas in the context of designing and implementing portals, i.e. specially designed services bringing information together from a set of heterogeneous services in a uniform way. The portal plays the role of a mediation service, encharged to integrate and mediate the exchange of information among the various services. Following (Brown, 2008), a portal is the ideal place where to locate the policy enforcement point which, together with a module for messaging, provides the interface to both users and the different services protected by their own authorization policy (see, Figure 1). Since the messaging module intercepts all messages flowing among the integrated services, the policy enforcement point can be invoked

Uttha W., Bertolissi C. and Ranise S..
Modeling Authorization Policies for Web Services in Presence of Transitive Dependencies.
DOI: 10.5220/0005548502930300
In *Proceedings of the 12th International Conference on Security and Cryptography* (SECRYPT-2015), pages 293-300
ISBN: 978-989-758-117-5
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

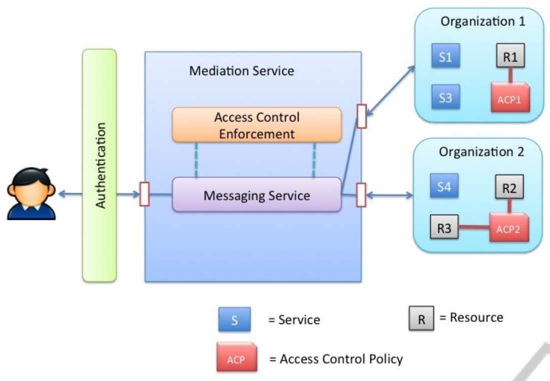

Figure 1: Access control with a mediation service.

whenever needed so as to enforce the appropriate access control policy and to record the transitive calls. For simplicity, we assume that the topology of the system (i.e. how the integrated services may invoke each other) is known and fixed.

Technically, we develop our approach by extending the ORganization Based Access Control (OrBAC) model (Y. Deswarte, 2009) with a delegation graph for transitive dependencies. Following a long tradition (see, e.g., (Li and Mitchell, 2003)), we use Datalog (Ceri et al., 1989) as the specification language in which to express our model. As a byproduct of choosing Datalog, our framework supports the automated analysis of execution scenarios through the invocation, off-the-shelf, of available Datalog engines (in our experiments, we use (Carbonnelle, 2014)). The main advantage of our analysis technique is twofold. First, there is no need to implement a prototype to experiment with the system behavior since it is possible to perform the simulation of the system from an abstract design before its deployment. Second, sometimes the code of some of the integrated services may not be easily accessible or may cost money to be invoked; it is indeed desirable to be able to build an abstract specification that can be used to predict (some of) the outcomes of the system without incurring in extra costs for testing. Additionally, the environment in which the various services will be run may not be easily accessible; thereby making the testing of the composed system very difficult, if possible at all.

The framework for modeling access control policies and the related automated analysis technique although a necessary first step in producing high quality SOA applications, are insufficient if not supported by adequate run-time support for the enforcement of policies in deployed systems. The **second contribution** of this paper is the description of an extension of the XACML architecture[1] to support the enforcement of access control policies in presence of transitive dependencies. The extension consists of incorporating the construction of the delegation graph in the policy information point of the XACML architecture for keeping track of delegated authorization rights in transitive chains of service invocations. We have implemented the architecture on top of the WSO2 platform,[2] an open source middle-ware platform for developing web service solutions, and we briefly report on our experience in using it with a case study.

Plan of the paper. Section 2 describes the access control problem with transitive dependencies and illustrates it on a case study adapted from (Fischer and Majumdar, 2008). Section 3 describes our formal modeling technique to solve the access control problem in presence of transitive dependencies. Section 4 shows how to use Datalog as a specification language for the model previously developed and explains how to invoke a Datalog engine to perform the (symbolic) simulation of scenarios. Section 5 describes the implementation of our solution and briefly discusses our experience with the case study introduced in Section 2. Section 6 discusses the related work and Section 7 concludes the paper. An extended version of this paper containing the complete Datalog model of the case study is available at http://pageperso.lif.univ-mrs.fr/~worachet.uttha/dl/secrypt2015-full.pdf.

2 CASE STUDY

To illustrate the transitive dependency problem, we have developed a case study inspired from (Fischer and Majumdar, 2008). We consider 4 collaborating yet independent medical services: the clinical management service managing the scheduling of patients, the laboratory information system tracking laboratory tests and their results, and the patient records service maintaining historical data about patients health. In addition, the web portal provides convenient web access to the previous three services. It does not store any confidential data locally. Instead, when the user requests a page, the portal makes service calls to the other services using the requesting user's attributes.

Each service has its own access control policy that is managed independently. Therefore, several different models of access control may have to cohabit and interact in the case of transitive calls (as detailed in Section 3). For such a kind of distributed, dynamic environments, the classical Role Based Access Control (RBAC) model shows its limitations. We

[1]http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[2]http://wso2.com

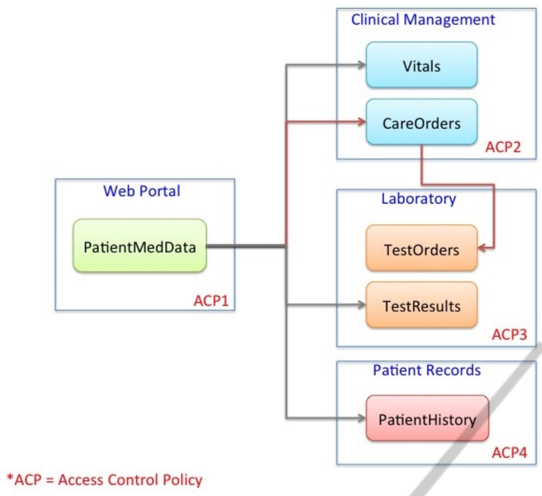

Figure 2: Topology of the case study in Section 2 with an example of transitive dependency (red path).

consider here a more general model called Category-Based Access Control (CBAC) model (Bertolissi and Fernández, 2014), where a category is any of several distinct classes or groups to which subjects may be assigned. CBAC is defined as a meta-model of access control that can be instantiated with several known models, e.g. with RBAC if the indirection between subjects and permissions is a set of roles, or Attributes Based Access Control (ABAC) if authorization privileges are based on user's credentials. In our case study, each service has a set of registered users to which categories are assigned according to e.g. their age, diploma, qualifications, etc. Permissions are then associated to each category. For instance, for *Clinical Management* we have the category *CM_Doctor* to which a user is assigned if she/he has a diploma in Medicine or his/her speciality is physician. The privileges associated to this category are consulting a file in vitals and care orders services.

The transitive dependencies problem in the medical scenario may occur *e.g.* when a user requests from the Web portal the access to some of his medical data such as the care orders prescribed by its physician. In this case, for answering the initial request, the service *Care Orders* may need to invoke the service *Test Orders* located in *Laboratory* to retrieve some details about a test ordered by the doctor. As each organization is protected by its own access control policy, the requester may have the right to make a call to the *Clinical Management* but this does not mean that he has the right to invoke the *Laboratory* to retrieve the test's results. We show this example of transitive call in red in Figure 2.

3 OUR FORMAL MODEL

In order to solve the problem mentioned in section 2, we will develop a formal model based on the Organization Based Access Control (OrBAC) model (Kalam et al., 2003).

In OrBAC it is possible to handle simultaneously several security policies associated with different *organizations* that can be seen as organized groups of activities or groups of services. Each security policy is defined for and by an organization and it is possible to handle simultaneously several security policies, since their specification is parametrized by the organization.

Although the OrBAC model can deal with the specification of policies for organizations or sub-organizations, the transitive access request is still not solved. In distributed environments, a subject may be recognized in an organization but it may not be known outside it. We will introduce the delegation of authority in order to overcome this issue. Moreover, in order to increase flexibility w.r.t standard OrBAC which structures subjects into roles as in RBAC, in our approach we structure the subjects in a general entity called category (as in (Bertolissi and Fernández, 2014)). Categories are associated to users on the base of the credentials they own, and permissions are assigned to each category.

Therefore, in our extended OrBAC model we have the following main entities:

- **Organization** (denoted org_0, org_1, $\dots$). can be seen as an organized group of activities (such as Clinical management, Laboratory and Patient Records in our case study) or it can be seen as an organized group of services (such as the Web portal in the case study).
- **Subjects and Categories.** The entity *Subject* (denoted s_0, s_1, $\dots$) is an active entity, (i.e. a user or an organization). Subjects that have the same attributes' value or satisfy same conditions belong to the same group, called *Category* (denoted cat_0, cat_1, $\dots$) and have the same permissions.
- **Objects and Actions**[3]. *Objects* (denoted o_0, $\dots$) are passive entities such as data files, patient's records, etc. *Action* entities (denoted act_0, act_1, $\dots$) contain computer actions such as "read", "write", "send" or "print".

The access control policy is modeled using the predicate *Permission(org, cat, act, o)* that specifies

[3]For easing the notation, we do not consider the abstraction of objects and actions as in the original OrBAC model, but this can be easily accommodated by adding 2 extra entities and the corresponding relations.

the permissions between categories, actions and objects, while the evaluation of requests is modeled using the predicate *Is_permitted(s, act, o)* that allows to derive permissions between subjects, actions and objects. The Subject-Category assignment is modeled using the predicate *Empower(org, s, cat)*.

In organization *org*, a subject *s* has a permission to perform an action *act* on an object *o* if (i) *s* is associated to a category *cat* in the organization *org* and (ii) the organization *org* grants the category *cat* the permission to perform the action *act* on the object *o*. The derivation of permissions is modeled by the following rule:

$$\forall org, \forall cat, \forall s, \forall o, \forall act, \\ Permission(org, cat, act, o) \wedge Empower(org, s, cat) \\ \rightarrow Is\_permitted(s, act, o) \quad (1)$$

Subject, action and object attributes are modeled by a set of binary predicates having the form *org_attribute(entity, value)*. For instance *CM_experience(David, 5)* means that David's work experience in *Clinical Management* is of 5 years. We can model the Subject-Category assignment according to the attributes of the subject as follows:

$$\forall org, \forall cat, \forall s, \\ Empower(org, s, cat) \leftarrow org\_attr_1(s, val_1) * \\ org\_attr_2(s, val_2) * \cdots * org\_attr_n(s, val_n) \quad (2)$$

Where * can be disjunction($\vee$), conjunction ($\wedge$) or a mixture of both.

As our system is distributed across several organizations, each having a different authorization domain, we have defined a delegation policy specifying the mapping of categories belonging to different authorization domains. This is formalized by a delegation graph in the form of Directed Acyclic Graph (DAG) that describes the way a category may be delegated (see Fig. 3). Each node represents a service authorization domain that can delegate categories (and thus permissions) to another. This supposes that an agreement on a set of categories that are allowed to be delegated has been previously reached between the different participants.

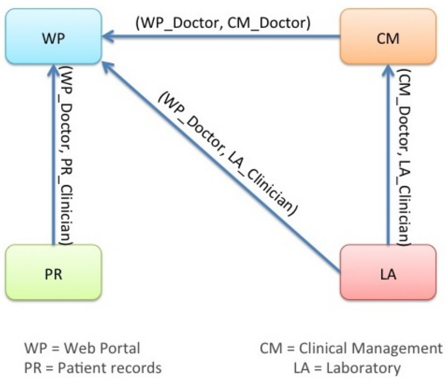

Figure 3: Delegation graph for the case study in Section 2.

The formal representation of the delegation graph is a logical predicate that takes four parameters *Delegate(org1, cat1, org2, cat2)* that means the category *cat1* from organization *org1* is mapped to the category *cat2* from the organization *org2*. The delegation graph is of crucial importance to determinate access request decisions in the case of transitive service invocation. The derivation of permissions in the case of a service invoking another service outside of its organization is modeled by the following rule:

$$\forall org1, \forall org2, \forall cat1, \forall cat2, \forall s, \forall o, \forall act, \\ Empower(org2, s, cat2) \wedge \\ Permission(org1, cat1, act, o) \wedge \\ Delegate(org1, cat1, org2, cat2) \\ \rightarrow Is\_permitted(s, act, o) \quad (3)$$

The rule above means that a subject *s* has a permission to perform an action *act* on an object *o* if (i) *s* is associated to a category *cat*2 in the organization *org*2, (ii) a category *cat2* from the organization *org2* is mapped to the category *cat1* in the organization *org1* and (iii) the organization *org*1 grants the category *cat*1 the permission to perform the action *act* on the object *o*.

4 AUTOMATED ANALYSIS OF EXECUTION SCENARIOS

Our approach can be validated through declarative programming, e.g., using a language such as Datalog based on facts, rules and queries with built-in support for backtracking.

Datalog. Facts are represented as *n*-ary relations of the form $fact(t_1, t_2, ..., t_n)$ where each t_i is either a constant or variable. Rules take the form $Head : -Body$, where the head is a fact and the body is a conjunction of facts, all of which need to be satisfied so that the head of the rule can succeed. A clause is either a fact or a rule. A Datalog program is a finite set of clauses. Datalog also allows one to ask questions, for instance $? - fact(t_1, t_2, ..., t_n)$ models a query for verifying that the considered fact holds. When implementing in Datalog, instead of specifying how to achieve a certain goal in a certain situation, we specify what the situation (rules and facts) and the goal (query) are and let the Prolog interpreter derive the answer true/false for us. For a thorough review on Datalog, we refer the reader to (Ceri et al., 1989).

Several techniques have been proposed for the efficient evaluation of Datalog programs; see (Abiteboul et al., 1995) for an introduction.

Example of Queries on the Case Study. We show now how the medical center case study can be specified and validated through Datalog. We consider the four organizations Web portal(wp), Clinical Management(cm), Laboratory(la) and Patient History(ph). User's attributes are represented by facts, such as $org(Bob, cm)$, $role(Bob, cm\_doctor)$, $speciality(Bob, dentist)$, specifying Bob's organization, role and speciality, respectively.

Each organization policy assigns subjects to categories (according to the subject's attributes). The Datalog rules allowing to define such assignments may be as following:

```
cat(wp,U,wp_nurse) :- role(U, nurse).
cat(cm,U,cm_senior_doctor) :-
     cat(cm,U,cm_doctor),
     experience(U,Exp), Exp >= 5.
```

The first rule means that a subject U is assigned to category *wp_nurse* in Web Portal if his/her role is nurse. The second rule means that for being assigned to the category *cm_senior_doctor* in *Clinical Management*, the subject must belong to the category *cm_doctor* and have a work experience greater or equal to 5 years.

In case of a transitive call, we need to determine if a service is required to invoke another service to satisfy a request or not. For that, we have defined the fact *depends_on(service1, service2)*. When *service1* $\neq$ *service2*, this means that *service1* uses *service2* to answer to the initial request, otherwise *service1* is independent. Moreover, we use the fact $belong\_to(service, organization)$ to determine to which organization a service belongs to. To answer the transitive request that we consider in Figure 2, the *careOrders_service* located in *Clinical Management* needs to invoke $testOrders\_service$ that belongs to *Laboratory*. This corresponds to the Datalog code below:

```
depends_on(careOrders_service,
 testOrders_service).
belong(careOrders_service, cm).
belong(testOrders_service, la).
```

We may have invocations that need only local evaluation and invocations that are global (inter-domain). In the last case we have to take into account transitive dependencies. This is done using recursion to compute the delegation chain. We have two main cases:

1. Service 1 invokes service 2 from another organization and service 2 is independent. Here, one step of delegation is required.
2. The transitive invocation chain is longer than 2 invocations, that is at least 2 steps of delegation are required. As an example, consider a service 1 that invokes a service 2 which in turn invokes an independent service 3, the three services being from different organizations. We need to consider this case because service 2 invokes service 3 on behalf of service 1 and not on behalf of itself, which is different from the first case.

We show below an example of access request evaluation where Bob from *Web Portal* tries to access a *Care Orders Service* file located in *Clinical Management*. Firstly the system determines Bob's organization and evaluates his attributes in order to resolve a category for Bob. Then, it checks the organization that owns the requested resource. If the subject's organization is different from the resource organization, a delegation step is needed in order to determine her/his permissions. If transitive dependencies are present, as it is the case in this example with *Care Orders Service* depending from *Test Orders Service*, the delegation chain is computed (see point 2. above). The query returns *True* if access rights are correctly propagated through the chain. It returns *False* if a denial of access is found somewhere in the path of involved services.

```
>>> request = is_permitted('bob','read',
'careOrders_service')
>>> print(len(request) > 0)
New fact : org('bob','wp')
New fact : role('bob','doctor')
New fact : cat('wp','bob','wp_doctor')
New fact : empower('wp','bob','wp_doctor')
New fact : depends_on('careOrders_service',
'testOrders_service')
New fact : belong('careOrders_service','cm')
New fact : !=('wp','cm') is True
New fact : delegate('cm','cm_doctor','wp',
'wp_doctor')
New fact : permission('cm','cm_doctor','read',
'careOrders_service')
...
New fact : is_permitted('bob','read',
'careOrders_service')
New fact : _pyD_query1()
True
```

Termination and Complexity. Since in our model, authorization queries are mapped to Datalog queries, we can study the problem of answering authorization queries by reusing well-known results (Ceri et al., 1989) about Datalog.

A first simple albeit desirable property of authorization queries is their finiteness, i.e. any authorization query admits finitely many answers. A well-known result in (Ceri et al., 1989) says that all the relations defined in a Datalog program are finite if the following conditions hold: (i) unit clauses (i.e. clauses composed of a single literal) are always ground; (ii) every variable that appears in the head of a rule also appears in a positive literal in the body of the rule, and (iii) no negative literal appears in the body of a rule. By construction, the Datalog programs specifying our models satisfy all these conditions.

Proposition 1. *In our model, authorization queries admit only finitely many answers.*

Termination of the process of answering queries is guaranteed by using a tabled logic programming algorithm (Chen and Warren, 1996), used by the pyDatalog (Carbonnelle, 2014) engine that we have used in our experiments.

Proposition 2. *In our model, answering authorization queries always terminates.*

Concerning complexity, recall that there are three types of complexity characterizations of answering Datalog queries (Ceri et al., 1989): *data complexity* (when rules are fixed, whereas facts and goal are an input); *expression complexity* (facts are fixed, rules and goal are an input); and *combined complexity* (rules, facts, and goal are an input). The rules in our Datalog program are all fixed since the topology of the system (together with the delegations) is fixed, the authorization policies are also given as they are not allowed to change during the operation of the system. What may change are the assignment of subjects-category and the assignment of subject-attributes that depend on the user's profile. This means that, in our context, the right complexity characterization to consider is data complexity. As a consequence, we can state the following result as a corollary of the polynomial time data complexity in (Dantsin et al., 2001).

Theorem 1. *In our model, answering authorization queries takes polynomial time.*

5 ARCHITECTURE AND IMPLEMENTATION OF THE ENFORCEMENT MECHANISM

So far, we have considered the modeling and analysis of authorization requests in web service applications at design time. In this section, we discuss the run-time enforcement of our access control model by extending the standard XACML architecture. We also discuss an implementation of the proposed enforcement mechanism on top of the WSO2 Identity Server,[4] which is part of an open source middleware platform supporting the development of SOA applications. The WSO2 Identity Server features, among many other things, attribute based access control via XACML. The development has been guided by the Datalog model presented above.

[4]http://wso2.com/products/identity-server

Architecture. Figure 4 shows the modules of the architecture interacting as follows. (1) The user's access request is intercepted by the Policy Enforcement Point (PEP); (2) the request and user's attributes are forwarded to the Policy Decision Point (PDP) which asks the Policy Information Point (PIP) for additional information about the subject. If the PDP determines from the policy and request that the user's organization is different from the resource organization, a delegation step is needed. In this case, (3) the PIP asks the delegation module, which contains the delegation graph and the history of previous delegations, to compute a category for the user in the new organization. The delegation module first checks the history of delegations of the user u making the request. If a category has been previously delegated to u in this organization, the module returns it to the PDP. Otherwise, it checks the delegation graph for the mapping of the user's category in the resource organization and returns it to PDP (4-5). The PDP loads the XACML policy of the organization owning the resource from the Policy Administration Point (PAP) (6-8). The PDP makes a decision and returns it to the PEP (9). If the answer is positive, the PEP forwards the request to the invoked service, otherwise the PEP throws an access denied exception (10).

Implementation. To implement the architecture in Figure 4, we have used the WSO2 Identity Server since it offers support for administering and enforcing XACML policies. In particular, it permits the customization of PAPs, PEPs, and PDPs. Policy administrators can specify access control policies by using an XML file or a graphic UI. All the modules and the interactions depicted in Figure 4 were already supported by the WSO2 Identity Server except for the additional module in the PIP for handling the delegation graph. We have implemented it in Java exploiting the JDBC API to maintain subjects and their attributes. This new module consists of 2 Java classes for a total of 350 lines of code.

Experience with the Case Study. The implementation described above have been used to animate the medical clinic portal described in Section 2. For this, we have implemented all services following the web service standard, i.e. UDDI for services discovery, WSDL for interface definitions and SOAP for invocations, XML as the format for exchanging messages. We have also implemented the PEP using the Java Servlet Filter for intercepting authorization requests, forward them to the PDP, wait for its decisions, and fi-

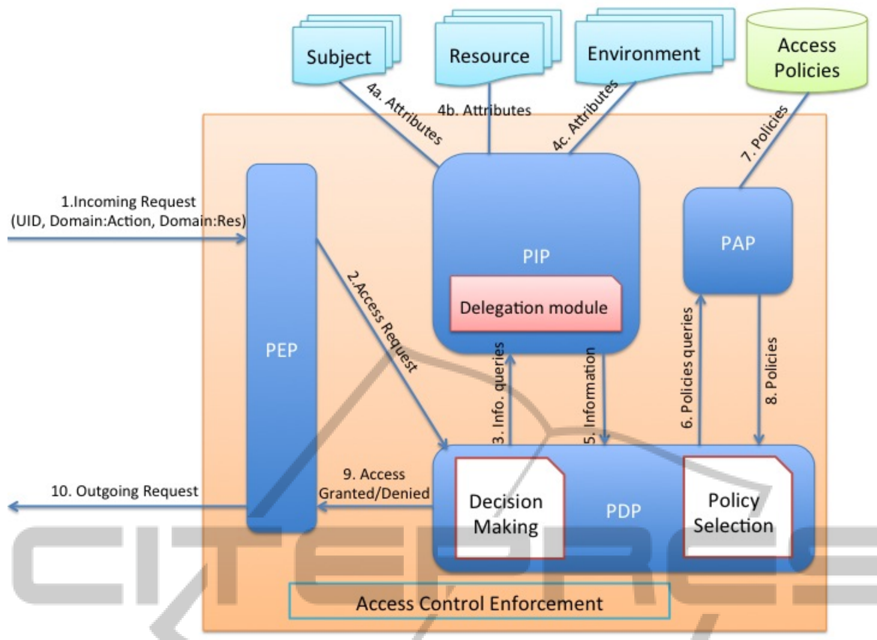

Figure 4: XACML Architecture with Delegation Graph Handling.

nally enforce them. We have then configured the PAP by specifying access control policies for all services via the WSO2 Identity Server interface.

We now describe how our implementation compute an answer to the (transitive) authorization request depicted in red in Figure 2. A user connects to Web portal and requests an access to Care order service. The Clinical Management's PEP intercepts the message and forwards it to the PDP on WSO2 Identity Server. The PDP computes an access decision and returns it to the PEP of the Clinical Management service. It may call the delegation module to obtain the delegation mapping from the Web portal to the Clinical Management service. If the request is accepted, the user can access to the Care order service hosted in Clinical Management. In the considered request, the Care Order service needs to invoke the Test order service in Laboratory to finish its task. Therefore, the Care Order service sends a request on behalf of the initial user to Laboratory and the Laboratory's PEP intercepts the request and sends it to the PDP. The PDP computes a decision as in point 3 and returns an answer to the Laboratory's PEP. If the access request is accepted, the user can access the Test order service and the invocation chain is finished. Otherwise, a security message is displayed to the user and the invocation chain is aborted.

All the process above, on a standard laptop, takes much less than a second to terminate. Indeed, the time for answering authorization requests increase with the length of the chain of service invocations. The WSO2 Identity Server features a caching mechanism for both attribute values and authorization decisions that permits to avoid repeating the same operation several times, thereby helping to improve the performances with long chains of service invocations.

6 RELATED WORK

The transitive access problem occurs frequently in Web Service based collaborative systems since each organization provides various services that are protected by its own security policy. As pointed out in (Li and Karp, 2007), the use of Federated Identity Management systems to solve transitive dependency is problematic. A solution to overcome this has been proposed in (Karp and Li, 2010) where the client needs to dialog directly with the policy engine. We choose instead to adopt the standard architecture where the user communicates directly with the service. This model is well suited to be implemented by using the available Identity Management Systems, such as WSO2. In (Chadwick et al., 2006) multiple policy domains and dynamic delegation of authority are considered. However, the authors do not specifically consider the problem of access request evaluation and access decision making in the case of transitive calls. The work in (Srivatsa et al., 2007) addresses the problem of access control for web service composition. The access policies are specified in Pure-Past Linear Temporal Logic (PPLTL) that allows to exploit the history of service invocations to make access control decisions. Unfortunately in practice the specification of policies in PPLTL is not very friendly for security designers. (She et al., 2013) and (Mecella et al., 2006) also discuss access control in web service composition. Nevertheless, their approach is different from ours. They consider the issue of service unavailability along a pathway to a target service, and they solve it by invoking dynamically alternative services belonging to different domains.

Several extensions of the OrBAC model have been proposed recently in order to specify security rules for intra- as well as inter-organizations. For instance (Y. Deswarte, 2009) have proposed a new access control framework for inter-Organizational Web services (PolyOrBAC). The authors model permissions, prohibitions and obligations in timed automata to verify properties such as reachability and correctness. They consider the case of a service requested remotely from a different organization. In this case, in order to authorize a user from a different domain to access a service, a particular role is associated to a virtual user, then a specific rule defined as a circumstance (context relation in OrBAC) is applied. However, they do not address the transitive access problem for dependent services, nor use requesters' credentials for computing the rights of access.

7 CONCLUSION

We have proposed a solution to the challenging problem of transitive dependencies in web service invocations by extending the OrBAC model with a cross-domain delegation graph (relation). Our model is flexible and dynamic since permissions are computed according to the value of users' credentials at the moment of the request, while in standard OrBAC, permissions are pre-assigned to users when the security policy is defined. By using Datalog as the specification language, the extended OrBAC model supports an automated analysis technique for executing scenarios before applications are deployed. We have also shown how access control policies can be enforced by extending the standard XACML architecture with a module handling the delegation graph.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Armando, A., Carbone, R., Compagna, L., and Pellegrino, G. (2012). Automatic Security Analysis of SAML-Based Single Sign-On Protocols.

Bertolissi, C. and Fernández, M. (2014). A metamodel of access control for distributed environments: Applications and properties. *Inf. Comput.*, 238:187–207.

Brown, P. (2008). *Implementing SOA: Total Architecture in Practice*. TIBCO Press Series. Addison-Wesley.

Carbonnelle, P. (2014). *pyDatalog*. https://sites.google.com/site/pydatalog/.

Ceri, S., Gottlob, G., and Tanca, L. (1989). What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166.

Chadwick, D., Otenko, S., and Nguyen, T. A. (2006). Adding support to xacml for dynamic delegation of authority in multiple domains. In *Communications and Multimedia Security*.

Chen, W. and Warren, D. S. (1996). Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43:43–1.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425.

Fischer, J. and Majumdar, R. (2008). A theory of role composition. In *IEEE Int. Conf. on Web Services*, pages 320–328.

Kalam, A., Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., and Trouessin, G. (2003). Organization based access control. In *4th Int. Ws. POLICY*, pages 120–131.

Karp, A. and Li, J. (2010). Solving the transitive access problem for the services oriented architecture. In *International Conference ARES*, pages 46–53.

Li, J. and Karp, A. H. (2007). Access control for the services oriented architecture. In *Proceedings of the 2007 ACM Workshop on Secure Web Services*, SWS '07, pages 9–17. ACM.

Li, N. and Mitchell, J. C. (2003). Datalog with constraints: a foundation for trust management languages. In *PADL'03*, pages 58–73.

Mecella, M., Ouzzani, M., Paci, F., and Bertino, E. (2006). Access control enforcement for conversation-based web services. In *15th Int. Conf. on WWW*, pages 257–266, USA. ACM.

She, W., Yen, I.-L., Thuraisingham, B., and Bertino, E. (2013). Security-aware service composition with fine-grained information flow control. *Services Computing, IEEE Transactions on*, 6(3):330–343.

Srivatsa, M., Iyengar, A., Mikalsen, T., Rouvellou, I., and Yin, J. (2007). An access control system for web service compositions. In *IEEE Int. Conf. on Web Services*, pages 1–8.

Y. Deswarte, A. A. E. K. (2009). *Poly-OrBAC: An access control model fior inter-organizational web services*. IGI-Global.