Toward an Abstract Language on Top of XACML for Web Services Security

Azzam Mourad[a], Hadi Otrok[b], Hamdi Yahyaoui[c] and Lama Baajour[a]
[a]Department of Computer Science and Mathematics,
Lebanese American University, Beirut, Lebanon
[b]Department of Computer Engineering,
Khalifa University of Science, Technology & Research, Abu Dhabi, UAE
[c]Department of Computer Science, Kuwait University, Kuwait

***Abstract*—We introduce in this paper an abstract language on top of XACML (eXtensible Access Control Markup Language) for web services security. It is based on the automatic generation of XACML security policies from abstract XACML profile(s). Our proposed approach allows first to specify the XACML profiles, which are then translated using our intended compiler into XACML security policies. The main contributions of our approach are: (1) Describing dynamic security policies using an abstract and user friendly profile language on top of XACML, (2) generating automatically the the XACML policies and (3) separating the business and security concerns of composite web services, and hence developing them separately. Our solution address the problems related to the complexity and difficulty of specifying security policies in XACML and other standard languages. We tested the feasibility of our approach by developing the library system (LB) that is composed of several Web services and applying/realizing our approach to enforce security.**
Keywords. Web Services Security; XACML; Security Policies; RBAC.

I. Introduction

Web service technology has been successful in making business applications available through the internet to a large number of users. It does not just widen the broad of applications accessibility but it is also a catalysis for collaboration between several distributed applications through the composition concept. Nevertheless, the successful deployment of this technology cannot hide the security breaches and threats [1] that a Web service can be exposed to. Enforcement of web service security is one of the most important duties, which the research community has to perform. This means the design and development of concepts, processes and tools that help in making Web services protected from malicious users. Instances of these security features can be the enforcement of user authentication, access control and data confidentiality in Web services.

However, due to the complexity of security policies and their composition of one or more combinations of rules, developers face several problems while building directly the security code. In this context, several standard languages have been proposed to specify policies and enforce web services security. The Security Assertion Markup Language (SAML) [2], WS-Security [3] and WS-XACML [4] are the most successful ones. The main problems with such language-based strategies is the difficulty and expertise needed for specifying the security policy rules, roles, procedures, targets, permissions, etc. Such a mechanism is cumbersome, error-prone and tedious. An example of XACML security policy is presented in Listing 1. This adds another requirement on the security developers to be expert in the business logic, security, policy specification languages and web services technologies. Such combination of requirements in one developer is by itself an open problem in the field of information system security [5].

In this paper, we propose an extension to the current XACML technology by introducing a new approach for the automatic generation of XACML security policy from XACML profiles. It is based on the elaboration of an abstract language on top of XACML for policy profile specification, which are then translated into XACML policies using our intended framework. The main contributions of our approach are threefold:

1) Describing dynamic security policies using an abstract and user friendly profile language on top of XACML, hence addressing the problem of the difficulty and expertise needed for policy specification. This will also support the friendly GUI with a language for policy composition analysis and verification.
2) Generating automatically the XACML policies from the XACML-Profile, hence addressing the problem of the complexity of security policy by specifying them using a standard language (i.e, XACML in our approach).
3) Separating the business and security concerns of composite Web services, and hence developing them separately.

To check the feasibility of our proposition, we developed a library system (LB) that is composed of web services. A RBAC (Role Based Access Control) model for the library system, which we called RBAC-LB, is elaborated. Afterwards, the web services that implement the security features are developed. Then, the XACML security policies that reflect the profile contents and integrate the security functionalities into the web services are built. The devised XACML policies realize the elaborated RBAC-LB model and provide authentication and access control features to the library system.

The rest of the paper is organized as follows. In Section II, we discuss the related work. Section III is devoted to the description of the library system architecture and its RBAC-LB model. Section IV is dedicated to the illustration of the proposed approach. In Section V, we illustrate the implementation of our propositions in a case study. Finally, we give some concluding remarks in Section VI.

II. Related Work

Web service security is one of the topics that attracted the attention of the research community. From the definition of standards to the publication of research papers, the goal is to provide policies and mechanisms for enforcing web services security. In this context, we first summarize and explore the advantages and limitations of the current standards and approaches for security policy description.

SAML [2] is an assertion language that is proposed by OASIS. Based on XML, it defines how to specify security credentials, which are represented as assertions. SAML can be used to manage secure sessions between organizations and can leverage several mechanisms such as basic password authentication, SSL and X. 509 certificates, etc. A security token is delivered to the requester after successful authentication. This security token allows granting certain permissions to the requester.

WS-Security [3] is a standard that is proposed by IBM, Microsoft, and Verisign. WS-Security is a means for using XML to encrypt and digitally sign SOAP messages. Another feature of WS-Security is that it allows exchanging security tokens for authentication and authorization of SOAP messages.

The Web Service eXtensible Access Control Markup Language (WS-XACML) [4] is proposed by OASIS as XML-based language to specify and exchange access control policies. WS-XACML is designed to define authorization policies for principals that are specified using XML.

Bhatti et al. [6] proposed X-RBAC: an XML-based RBAC policy specification framework for enforcing access control in dynamic XML-based Web services. The specification uses representations of users, roles and permissions. The two main components of the proposed framework are: the XML and the RBAC processors. The XML processor is implemented in Java using Java API for XML Processing (JAXP). Some modules have the duty to get the DOM instance of parsed XML documents and forward them to the RBAC Processor. The RBAC module is responsible for administration and enforcement of the policy according to the supplied policy information.

Damiani et al. [7] proposed a design of a Web service architecture for enforcing access control policies. They also provide an example of implementation based on the WS-Policy [8], [9] as access control language. The main components of the proposed architecture are: Policy Administration Point (PAP), Policy Evaluation Point (PEP) and Policy Decision Point (PDP). The PAP module is a policy repository that provides an administrative interface for inserting, updating, and deleting policies. The PEP module realizes the enforcement of the policies returned by the PAP module. The access request is granted if at least one policy is satisfied; the access is denied otherwise. A PDP module is the interface between the service and the enforcer module. It is responsible for taking final access control decisions based on the input from the PEP module.

Bertino et al. [10] proposed a RBAC-WS-BPEL framework for defining authorization policies and constraints for WS-BPEL business processes. WS-BPEL is a language for composing Web services. To specify authorization policies, the authors used XACML. They introduce the Business Process Constraint Language (BPCL), which can be used to specify authorization constraints. The users are associated with roles as done in Role Based Access Control (RBAC) models.

Ramesh proposed in his Masters thesis a study that evaluates the advantages/disadvantages of integrating XACML with existing .NET security [11]. In his study he evaluated the effect of XACML integration on the existing rules engine and the system's performance. He elaborated his idea on IBIS (Infinite Business Information System) website having security implemented in two modes: XACML mode and non-XACML one. Based on a comparison between the above two implementations, he came up to a number of significant conclusions that highlights the ability of XACML in overcoming language shortcomings, its applicability to several resources, and reusability of its built-in functions by developers.

The aforementioned related work support our claim about the need to have standard languages for the description and specification of security policies. However, the main limitations of the proposed approaches is the difficulty and expertise needed for specifying the security policy rules, roles, procedures, targets, permissions, etc. Such a mechanism is cumbersome, error-prone and tedious. To the best of our knowledge, none of them address such problem, which is the main contribution of our propositions.

On the other hand, Sanchez et al. [12] proposed a user friendly approach that uses high level templates to specify security policies. They selected Microsoft Infopath tool to design those templates. In their approach, the administrator identifies the elements of the policy in an XML schema. They claim that this XML document is then transformed to XACML policy using XSL transformation by respecting XACML format and namespaces. The output of XSL transformation is a fixed policy structure built using xsl:element and xsl:attribute. Once the transformation is complete, it needs to be applied to every InfoPath-generated document. However, they didn't provide any detail about their methodology and practical application of their approach. Their initiatives is limited to a graphical interface on top of XACML in the Microsoft Infopath tool environment. We are addressing the same problem in our approach, however we rely on elaborating a well-defined language, compiler and development environment on top of the XACML technology.

XACML.NET is a user friendly tool that allows the evaluation and editing of XACML policies. It was developed by Diego Gonzales, a professional software developer, in April

2004. XACML.NET is an implementation of the XACML Policy Decision Point (PDP) component, in a pure .Net open source code and specifically in C# language. It conforms to XACML 1.0 specification, ratified by OASIS standards organization. This tool contains four assemblies but the main two are the ”Core” and the ”Control Center”. The ”Core” is the main code that comprises the set of classes responsible for loading the policies and requests, in its XML representation, and accordingly the generation of a given response. The ”Control Center” is the Graphical User Interface (GUI) that is used for the creation, editing and evaluation of the XACML policies and requests. Several releases of this software followed and version 0.7, issued in March 2005, was the last of all.

UMU-XACML-Editor [13] was developed by University of Murcia (UMU) to provide support for the creation and validation of XACML access control policy files. It is a graphic policy definition editor implemented in Java language and based on the XACML standards. This editor provides a syntax-directed manner to construct a correct XACML policy. It adopts a tree-based user interface design as to make policy integration seamless from a user perspective. Thus, it allows users to edit and save policies in a friendly user interface with the constraints of XACML 2.0 specifications that were defined by OASIS.

III. RBAC-LB: An Authorization Model for a Library System Web Services

In this section, we focus on describing our library system and elaborating its corresponding role based authorization model by following our proposed approach. We first present an overview of the library system and describe the proposed RBAC-LB model. Afterwards, we present aspects and patterns realizing the proposed approach and model.

A. Library System Overview

Our system (illustrated in Figure 1) is a Library Management System, in particular, the Circulation division in the Library. We developed four of the main services of Circulation division in addition to a graphical user interface to interact with the user, get the input and show the results. A user can access one of the services in the system's main page. First, the View Info service allows him/her to view his/her personal information (Name, ID, Date of Birth, Telephone Number, User Type, Address, and Email). Second, Add/Delete library User service allows him/her to add or delete a user to/from the library. Third, Add/Delete Library Item is a service which allows a user to add or delete a new library item to/from the library. Finally, a user may access the Search/Borrow A Library Item service, which allows him/her to search for a library item and borrow it if its available in the library. Each one of the users who can access the system has a record in the database. In other words, each user has an ID and a Password stored in the database in addition to his/her personal information. Moreover, the system database contains a record for each library item which holds information for the following fields: ISBN, type, subject, title, author, and status.

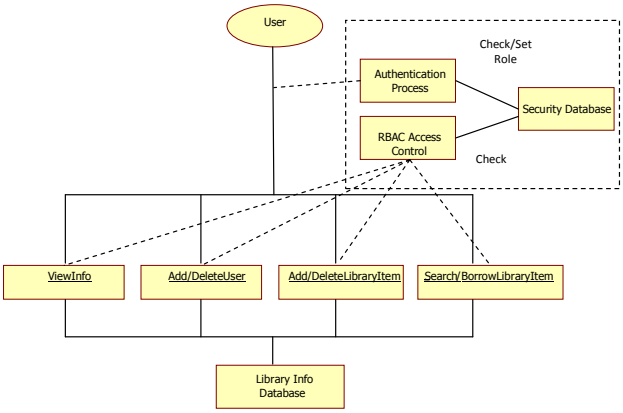

Figure 1. Library System Architecture

We extended our library system by elaborating authentication and access control features. The extension is explored in Figure 1.

B. RBAC-LB Model Definitions

This model inherits all the components of the traditional RBAC model such as: users, roles, permissions, role hierarchies, user-role assignment, and role-permission assignment relations. Users are assigned to roles and roles are assigned to permissions. An RBAC permission represents the ability to access a certain system's service. A user is permitted to execute a service activity if he/she is assigned to a role that has the permission to perform that activity. RBAC roles are structured in a hierarchy. Role hierarchies define inheritance relations among the roles in terms of permissions and user assignments. If role r_1 inherits role r_2, then all permissions of r_2 are also permissions of r_1 and all users of r_1 are also users of r_2. More detail about describing RBAC model for web services is presented in [10]. The following is the list of notations:

A: is the identifier of an activity (e.g. Add a new book to the library shelves).

Action: is the type of action that can be performed on activity A.

R: is the set of roles (e.g.: Admin, Staff, Student).

U: is the set of potential users.

P: is the set of permissions, i.e. execution of an activity (e.g. execution of add/delete library user).

To render our RBAC model specification open to future extensions, we will start with a general definition of the main components of the RBAC model.

Definition 1: RBAC-LB Permission: Let LB be our system. An RBAC-LB permission is a tuple (A_i, Action) where A_i is the identifier of an activity in LB and *Action* identifies the type of action that can be performed on activity A_i. For example, the tuple (Add library user, execute) allows the authorized user to run the “Add library service” that is provided by the Library Management System.

Definition 2*: RBAC-LB Role:* An RBAC-LB role r is a set of attribute conditions r= $\{ac_i \mid ac_i$=$AttrName_i$ *op* $AttrValue_i\}$, where $AttrName_i$ identifies a user attribute name, *op* is a comparison or a set operator, and $AttrValue_i$ is a value, a set, or a range of attribute values.

Two roles r and r' might be identified by the same set of attribute names. However, it is a must that at least one of the values of the attributes of r and r' must be different. A user can be assigned to only one role while two users identified by the same attributes with the same values are assigned to the same role since we assume that a set of attribute conditions uniquely identifies a role.

Definition 3*: RBAC-LB Role Hierarchy:* Let R be a partially ordered set of roles. A role hierarchy defined over R is the graph of the partial-order relation between the roles in R. If r, $r_0 \in R$ and $r < r_0$, then we say r_0 dominates r. For instance, our library system consists of eight different roles. The most senior role is the $Admin$ which dominates the roles $Staff$ and $Student$. $Staff$, in turn, dominates $Faculty$ and $AdminstrativeStaff$, and $Student$ role dominates $UndergradStudent$, $GradStudent$, and $Alumni$. For example, a user that wants to be assigned to the $Admin$ role must provide digital credentials containing an $Employment$ attribute to $Admin$. Figure 2 and Table I illustrate the role hierarchy for the library management system.

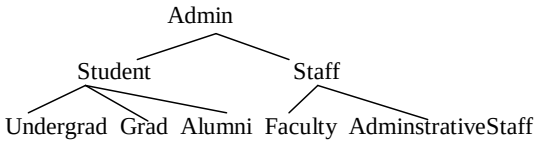

Figure 2. RBAC-LB Role Hierarchy

Table I
RBAC-LB ROLE HIERARCHY

R	
Admin	{ Employment= Admin, ID= integer of 9 digits, Password= a string of at most 9 characters}
Staff	{ Employment= Staff, ID= integer of 9 digits, Password= a string of at most 9 characters}
Student	{Employment= Student, ID= integer of 9 digits, Password= a string of at most 9 characters}

Definition 4*: RBAC-LB User-Role Assignment Relation:* Let U be the set of all potential users and R be a partially ordered set of roles. The RBAC-LB user assignment relation is the set of tuples UA= {(u,r) ∈ U x R | $\forall\ ac_i = AttrName_i$ *op* $AttrValue_i \in$ r, $\exists\ attr_j \in$ CredSet$(u)^1$ | $attr_j$= $AttrName_i$ $\bigwedge\ ac_i$ is evaluated to true according to value of $attr_j$}. As for the library system, all university members can be considered as potential users and the set of roles is R={ *Admin*, *Staff*, *AdminstrativeStaff*, *Faculty*, *Student*, *Alumni*, *Grad*, *UnderGrad*}. Assigning users to roles results in a set of tuples which define the RBAC-Library user assignment relation. For example, in our library system the set of attribute conditions for the Admin role is r= {Type= "Admin", ID= an integer of 9 digits, Password= a string of at most 9 characters}; thus a credential set of the user u={Type= "Admin", ID= "100100100", Password= "100100"} will be evaluated as true and u is assigned to $Admin$.

Definition 5*: RBAC-LB User-Permission Assignment:* Let P be the set of permissions of the form (A_1, A_2) where A_1 is the activity supported by the system, and A_2 is the action to be executed. Moreover, let RP be the set of permission role assignment. Thus, the RBAC- LB user- permission assignment relation is the set of tuples UP= {(u,p) ∈ U x P | ∃ (u,r) ∈ UA | (r,p) ∈ RP}. For instance, a permission of adding or deleting a library user is, of the form ("Add/Delete Library user", execute), assigned to $Admin$ by the RP relation. Thus a user u can add/delete a library user only if he is assigned first to $Admin$.

IV. APPROACH DESCRIPTION

XACML is one of the promising policy enforcement languages that can ensure web services security. XACML [4], in addition to other standard languages [2], [3], are very useful for the organized description of composed security policies. They allow to avoid the ad-hoc description of security rules and specify them in XML-based document(s). On the other hand, they still suffer from the complexity, difficulty and expertise needed for policy specification, which limits its potential in supporting the security of web services. A developer need to reflect the organizational security requirements using complex XACML (or any other language) constructs where rules, roles, targets, subjects, permissions and legal actions are defined. Such a mechanism is cumbersome, error-prone and tedious.

In this context, we present in this section a methodology to develop security policies using a XACML Profile language and editor. This methodology will allow a non-expert business oriented people to integrate security into their web services. Such approach contributes in solving the aforementioned problems related to the specification and integration of security policies. It is composed of two related phases. The first one provides a user-oriented XACML Profile language and editor that are used to develop XACML policies. The second phase is an XACML generator that translates user defined commands to XACML ones. The aforementioned editor is based on an abstract language that uses high level commands to describe a certain policy. After defining the policies, our language compiler verifies the syntax correctness and translates the high level commands to XACML ones.

The proposed approach is depicted in Figure 3. It illustrates the XACML profiles that describe the security model using the proposed XACML Profile Language. These profiles are then compiled by the XACML Generator to produce the XACML policies. A compiler for XACML Profile Language is integrated in the XACML Generator. Afterwards, the generated XACML security policies are enforced on a select set of web services. Examples of XACML security policies are presented in Section V. The development of the intended framework is still in progress.

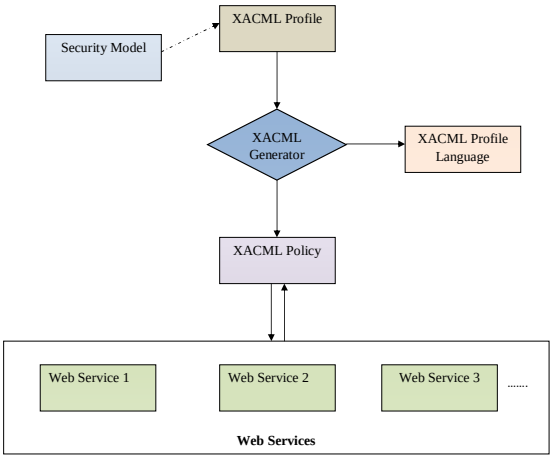

Figure 3. Approach Schema

In the following, we present an informal description of the required constructs in the intended XACML-Profile language.

PolicySet: composed of ID, CombiningAlgorithm, Target, and one or more Policy.
Policy:composed of ID, CombiningAlgorithm, Target and RuleSet.
RuleSet: composed of one or more Rule.
Rule: composed of ID and Target.
Target:composed of SubjectSet, ResourceSet and ActionSet.
SubjectSet: composed of one or more Subject.
Subject: composed of ID and Attribute.
ResourceSet:composed of one or more Resource.
Resource: composed of ID and Attribute.
ActionSet: composed of one or more Action.
Action: composed of ID and Attribute.

A compiler is under development to integrate the aforementioned constructs in the XACML-Profile language. A framework environment and a graphical user interface will be developed as alternative for the developer to specify the security policy. Once done, the XACML-Profile policy will be compiled into XACML policy, which in its turn will be applied on the web services.

V. Case Study: Enforcement of the XACML RBAC-LB Model in the LB

In this section, we present the implementation of the XACML RBAC-LB model that illustrates part of the procedures and mechanisms described in our proposed approach. It allows the enforcement of authentication and access control features in the library system web services.

A. RBAC-LB XACML Specification

Listing 1 outlines a summary of XACML- based access control policy for a Library Circulation System. Due to space limitation, we included in the listing the role and permissions of the Admin. The others are set in similar way. First, the roles are defined. A general role (root of the hierarchy) is denoted by Admin. It has 2 sub- roles, Staff and Student, and as an Admin he is able to give others the permission to perform any action on any resource. Staff are assigned to RPS:Staff: role, and students are assigned to RPS: Students: role. Each role has a corresponding permission policy which defines the set of permissions assigned to it. For instance, RPS:Student:role includes the permission policy which allows the student to view personal information and search / borrow a certain library item.

The application starts by receiving the user login info. After receiving the input, the authentication web service gets invoked to ensure that the user has a valid username and password. If he is not an authenticated user, the application returns an error notifying the client of the authentication failure. On the other hand, if the user is authenticated, the process will continue to assign the client input to the access control request message. Then the XACML policy is checked to determine the client level of permission and to check if the client has the right to see the requested service. If the access is denied, the process pops up a message to the user to inform him about the access control restrictions.

Listing 1. XACML-based Access Control Policy Specification for a Library System

```
<PolicySet>
<!--Defining the role policy set for the Admin-->
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:
     os" PolicyId="RPS:Admin:role" PolicyCombiningAlgId="
     policy-combine:permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="function:anyURI-equal">
          <AttributeValue DataType="xml:anyURI">Admin</
               AttributeValue>
          <SubjectAttributeDesignator AttributeId="role"
               DataType="xml:anyURI"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
     <Resources>
        <AnyResource/>
     </Resources>
       <Actions>
         <AnyAction/>
       </Actions>
  </Target>
  <!-- Use permissions associated with the Admin role -->
  <PolicyIdReference>PPS:Admin:role</PolicyIdReference>
</Policy>

<!-- Defining the permissions policy set for the Admin-->
  <Policy PolicyId="PPS:Admin:role" RuleCombiningAlgId="
       rule-combine:permit-overrides">
  <!-- Permission to Add a library item -->
  <Rule RuleId="Permission:to:add:library:item" Effect="
       Permit">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="function:string-equal">
          <AttributeValue DataType="xml:string">item</
               AttributeValue>
          <ResourceAttributeDesignator AttributeId="
               resource:resource-id" DataType="xml:string"/>
        </ResourceMatch>
      </Resource>
    </Resources>

    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
```

```
          <AttributeValue DataType="xml:string">add</
               AttributeValue>
          <ActionAttributeDesignator AttributeId="action:
               action-id" DataType="xml:string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  </Rule>

  <!-- Permission to delete a library item -->
  <Rule RuleId="Permission:to:delete:library:item" Effect="
       Permit">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="function:string-equal">
          <AttributeValue DataType="xml:string">item</
               AttributeValue>
          <ResourceAttributeDesignator AttributeId="
               resource:resource-id" DataType="xml:string"/>
        </ResourceMatch>
      </Resource>
    </Resources>

    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="xml:string">delete</
               AttributeValue>
          <ActionAttributeDesignator AttributeId="action:
               action-id" DataType="xml:string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  </Rule>
  <!-- Include permissions associated with staff role -->
  <PolicyIdReference>PPS:staff:role</PolicyIdReference>
  <!-- Include permissions associated with student role -->
  <PolicyIdReference>PPS:student:role</PolicyIdReference>
</Policy>

<!--Defining the role policy set for the Staff-->
...
<!-- Defining the permissions policy set for the Staff-->
  ...
  <!-- Permission to Add a library item -->
  ...
  <!-- Permission to delete a library item -->
  ...
  <!-- Include permissions associated with student role -->
  ...

<!-- Defining the role policy set for the student-->
...
<!-- Defining the permissions policy set for the student-->
  ...
  <!-- Permission to view his personal information -->
  ...
  <!-- Permission to search for a library item -->
  ...
  <!-- Permission to borrow a library item -->
  ...
</PolicySet>
```

B. Discussion and Experimental Results

Based on the proposed approach presented in Section 3, we applied the XACML policy in Listing 1 on the library system web services. Verifying the successful deployment of the RBAC-LB security features in the original code of the Library system application has been done through extensive testing. Other testing and simulation methods could be applied. Additional efforts have been spent on verifying that the original functionalities of the applications have not been altered.

VI. Conclusion

We presented in this paper a new approach for the automatic generation of XACML policies from the XACML profiles. Our proposition is based on the elaboration of an abstract language on top of XACML for policy profile specification, which are then translated into XACML policies using our intended framework. The experiments resulting from developing the XACML profiles and policies of the RBAC-LB model and then applying them on the library system web service, explored the feasibility and appropriateness of our propositions. They also illustrate the successful integration of authentication and access control features in the LB system.

References

[1] K. N. Computing, "XML and Web Services: Message Processing Vulnerabilities," http://www.webservicessummit.com/Articles/MessagingThreats.htm.

[2] B. Lockhart and al., "OASIS Security Services TC (SAML)," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

[3] B. Atkinson and al., "Web services security (WS-Security)," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[4] T. Moses, "OASIS eXtensible Access Control Markup Language (XACML), OASIS Standard 2.0," http://www.oasis-open.org/committees/xacml/.

[5] A. Mourad, M.-A. Laverdière, and M. Debbabi, "A high-level aspect-oriented based framework for software security hardening," *Information Security Journal: A Global Perspective*, vol. 17, no. 2, pp. 56–74, 2008.

[6] R. Bhatti, J. Joshi, E. Bertino, and A. Ghafoor, "Access Control in Dynamic XML-Based Web-Services with X-RBAC," in *Proceedings of the International Conference on Web Services (ICWS'03)*, 2003, pp. 243–249.

[7] C. A. Ardagna, E. Damiani, S. D. C. di Vimercati, and P. Samarati, "A Web Service Architecture for Enforcing Access Control Policies," *Electronic Notes Theoretical Computer Science*, vol. 142, pp. 47–62, 2006.

[8] J. Schlimmer, "Web Services Policy Framework (WS-Policy)," 2004, http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/.

[9] P. Nolan, "Understand WS-Policy processing," IBM Corporation, Tech. Rep., 2004.

[10] F. Paci, E. Bertino, and J. Crampton, "An Access-Control Framework for WS-BPEL," *International Journal of Web Services Research*, vol. 5, no. 3, pp. 20–43, 2008.

[11] R. et al., "Integration of XACML with .NET to Implement Secure Enterprise Web Applications," in *Project Report*, 2005.

[12] M. Snchez, G. Lpez, A. F. Gmez-Skarmeta, and scar Cnovas, "Reverte: Using Microsoft Office Infopath to Generate XACML Policies," in *Proceedings of the International Conference on Security and Cryptography(SECRYPT 2006)*, Setubal, Portugal, August 2006.

[13] P. G. Morcillo and A. J. Lzaro, "UMU-XACML-Editor. http://xacml.dif.um.es."