Generating XACML Enforcement Policies for Role-Based Access Control of XML Documents

Alberto De la Rosa Algarín[1(✉)], Timoteus B. Ziminski[1], Steven A. Demurjian[1], Yaira K. Rivera Sánchez[1], and Robert Kuykendall[2]

[1] Department of Computer Science and Engineering, University of Connecticut, Storrs, USA
{ada, tbz, steve, yaira}@engr.uconn.edu
[2] Department of Computer Science, Columbia University, New York, USA
rrk2l36@columbia.edu

Abstract. Ensuring the security of electronic data has morphed into one of the most important requirements in domains such as health care, where the eXtensible Markup Language (XML) has been leveraged via standards such as the Health Level 7's Clinical Document Architecture and the Continuity of Care Record. These standards dictate a need for approaches to secure XML schemas and documents. In this paper, we present a secure information engineering method that is capable of generating eXtensible Access Control Markup Language (XACML) enforcement policies, defined in a role-based access control model (RBAC), that target XML schemas and their instances, allowing instances to be customized for users depending on their roles. To achieve this goal, we extend the Unified Modeling Language (UML) with two new diagrams: the XML Schema Class Diagram, which defines the structure of an XML document in UML style; and, the XML Role-Slice Diagram, which defines roles and associated privileges at a granular access control level. We utilize a personal health assistant mobile application for medication and chronic disease management to demonstrate the enforcement component of our work.

Keywords: Security and policy modeling · Security policies · XML · XACML · Role-based access control

1 Introduction

Securing sensitive and private information has evolved into a needed requirement in domains such as healthcare informatics, where the daily workflow depends on the secure management and exchange of information, often in time-critical situations. In healthcare informatics, the eXtensible Markup Language (XML) is used for data and information exchange across heterogeneous systems via XML standards such as Health Level Seven's clinical document architecture (CDA) [10] for health information exchange and the Continuity of Care Record (CCR) for capturing clinical patient data. In such settings, security protection must be insured so individuals have

K.-H. Krempels and A. Stocker (Eds.): WEBIST 2013, LNBIP 189, pp. 21–36, 2014.
DOI: 10.1007/978-3-662-44300-2_2

the appropriate credentials to access all of the required data (clinical, genomic, other phenotypic, etc.) in accordance with the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [1]. For the purposes of our work, we propose a secure information engineering method using the Unified Modeling Language (UML) to define and enforce XACML role-based access control (RBAC) security policies that allow XML schemas to be controlled and instances customized based on role, time, and usage.

The main objective of this paper is to create security policies defined and realized in XACML that target XML schemas and their instances to provide granular document-level security. The enforcement of these policies permits document instances to look different to authorized users at specific times based on the user's role. In contrast to the general research done in XML security, which typically embeds security policies as part of the XML schema's definition, our approach allows policies to be evolved and applied to an application's XML instances without changes to instances and schemas. This approach results in a separation of concerns for facilitating security policy evolution without impacting XML instances.

To support this process, we have defined a security framework for XML in prior work [9] as shown in Fig. 1. The general approach is to have a set of XML schemas corresponding to an application (middle right in Fig. 1), which will be instantiated for the executing application (bottom right of Fig. 1). From a security perspective, our intent is to insure that when users attempt to access the instances, that access will be customized and filtered based on their defined user role and associated security privileges (role restricted, or RR, bottom left of Fig. 1). To achieve this in a secure information engineering context, the framework in Fig. 1 contains two new UML diagrams: the XML Schema Class Diagram (XSCD) that represents the structure of an XML document in UML style design artifacts; and, the XML Role-Slice Diagram (XRSD) that supports RBAC through the definition of granular access to XML schemas (and associated instances) based on role.

This paper to extends our earlier work [9] by concentrating on the left hand side of Fig. 1 (the XACML Policy Mapping box) for the definition and generation of XACML security policies and their enforcement at the runtime level on XML instances to insure that filtered, correct, and required data is securely delivered. The emphasis of this paper is on the generation of XACML security policies from XRSD diagrams that allow for the enforcement of those policies at runtime, which changes to the policy able to be made so that there is no impact on the original XML schema and its existing instances. Our proposed security framework will be applied to the health care domain, specifically to the CCR schema, using a case study of a mobile health application, Personal Health Assistant (PHA), for general health management.

The remainder of this paper is organized as follows. In Sect. 2, we present related work on XML security and access control, focusing on the approaches of embedded security and general access control. Section 3 provides background information on NIST RBAC, XML and XACML, the CCR standard; and a review of the key facets of our XML security framework that are needed to explain XACML policy generation. In Sect. 4, we present the mapping process and rules that generate XACML policies process from the XRSDs of a given XML schema, including an algorithm. In Sect. 5, we demonstrate the XACML policy interplay and enforcement with PHA, describing

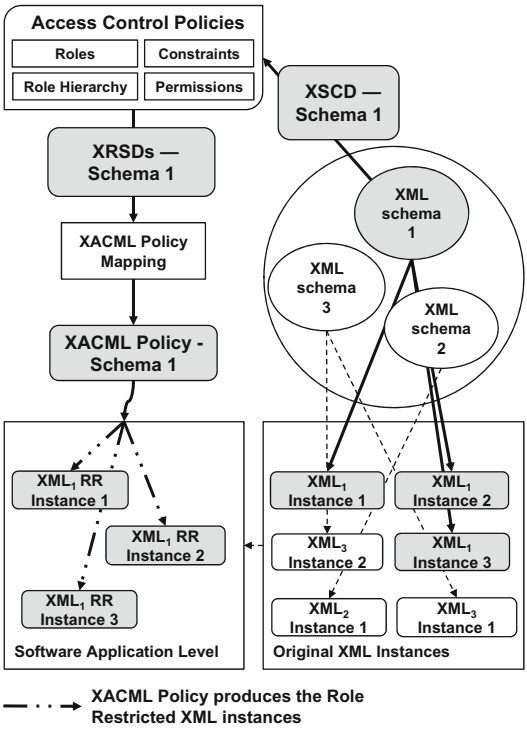

Fig. 1. Security framework for XML.

in detail the way that the patient and provider use them for information sharing and the achievement of enforcement. We finish the paper by offering concluding remarks and ongoing work in Sect. 6.

2 Related Work

The work of [5] presents an access control system that embeds the definition and enforcement of the security policies in the structure of the XML documents in DTDs in order to provide customizable security. This provides a level of generalization for documents that share the same DTD, similar to our work where security policies act against XML schemas to control XML instances. Two differences are: their work targets outdated XML DTD's while ours utilizes schemas, and their polices are embedded into both DTD and instance, requiring changes to instances when policies change; our work allows changes with no impact on instances.

Another effort [6] details a model that combines the embedding of policies and rewriting of access queries to provide security to XML datasets. The XML schema is extended with three security attributes: access, condition, and dirty. While this work is similar to our work by targeting security in XML instances via policies, it differs by requiring changes to instance when the policy is modified and does not consider XML document writing (see Sect. 5.3).

Efforts by [2, 3] present Author-X, a Java-based system for DAC in XML documents that provides customizable protection to the documents with positive and negative authorizations. Author-X employs a policy-based DTD document that prunes an XML instance based on the security policies, which is similar to our approach, but focuses on discretionary access control where we focus on RBAC. The work of [14] considers the scenario of a federated access control model, in which the data provider and policy enforcement are handled by different organizations. This approach relates to ours with regards to the separation of the security policies from the data to be handled, but differs in the specifics of where the policies' details are stored.

The work of [13] has presented a model consisting of access control policies over outmoded DTD's with XPath expressions to achieve XML security. Their model is similar to ours, as it aims to provide different authorized views of an XML document based on the user's credentials. However, the significant difference is that this approach combines query rewriting and authentication methods, whereas our approach can be applied to any non-normative XACML architecture (having a policy enforcement point) for both reading and updating, as well as XPath or XQuery queries.

The work of [15] presents an approach of supporting RBAC to handle the special case of role proliferation, which is an administrative issue that happens in RBAC when roles are changed, added, and evolve over time, making security of an organization difficult to manage. Our approach doesn't address role proliferation; however, by separating our security into an XACML policy, we do insulate role proliferation from impacting an application's XML schemas and instances.

3 Background

The NIST RBAC [12] standard has permissions assigned to roles which are assigned to users. NIST RBAC has four reference models. In $RBAC_0$, policies can be defined at the role level instead of the individual level. In $RBAC_1$, parent roles can pass down common privileges to children roles. In $RBAC_2$ the separations of duty (SoD) and cardinality constraints are provided, ensuring the role that grants permissions (authorization role) exists in a different entity to the other roles. The last reference model, $RBAC_3$, introduces the concept of sessions (lifetime of a user, role, permission and their association for a runtime setting).

XML facilitates information exchange across systems by providing a common structure to information that is hierarchically structured and tagged, where tags can be used to represent the semantics of the information. XML offers the ability to define standards via XML schemas, which serve as both the blueprint and validation agents for instances to comply and be used for information exchange purposes. The Continuity of Care Record (CCR) standard allows the creation of documents with patient information (demographics, social security number, insurance policy details, medications, procedures, etc.) and a common structure for uniform information exchange across institutions. The CCR schema contains elements for virtually all health information items, and is represented with extended granularity for better detail keeping. For example, Fig. 2 shows a subset of the official CCR schema

```
<xs:complexType name="StructuredProductType">
  <xs:complexContent>
    <xs:extension base="CCRCodedDataObjectType">
    <xs:sequence>
      <xs:element name="Product" maxOccurs="unbounded">
        <xs:complexType>
        <xs:sequence>
          <xs:element name="ProductName"
           type="CodedDescriptionType"/>
          <xs:element name="BrandName"
           type="CodedDescriptionType" minOccurs="0"/>
          <xs:element name="Strength" minOccurs="0"
           maxOccurs="unbounded">
          <xs:complexType>
          <xs:complexContent>
          <xs:extension base="MeasureType">
          <xs:sequence>
            <xs:element name="StrengthSequencePosition"
             type="xs:integer" minOccurs="0"/>
            <xs:element name="VariableStrengthModifier"
             type="CodedDescriptionType" minOccurs="0"/>
          </xs:sequence>
          </xs:extension>
          </xs:complexContent>
          </xs:complexType>
          </xs:element>
          <xs:element name="Concentration" minOccurs="0"
           maxOccurs="unbounded">
          <xs:complexType>
          <xs:complexContent>
```

Fig. 2. Segment of the CCR schema's StructuredProductType.

corresponding to the complexType element StructuredProductType, which is utilized to represent medications and their attributes. This StructuredProductType is used throughout this paper to explain the modeling and policy generation in an example health care scenario.

Our prior work has defined new UML security diagrams for supporting RBAC [16] via the UML meta-model. Using this as a basis, we have extended this work to define two new UML artifacts [8, 9]: the XML Schema Class Diagram (XSCD), which contains architecture, structure characteristics, and constraints of an XML schema; and the XML Role Slice Diagram (XRSD), which has the ability to add permissions to the various elements of the XSCD. Figure 3a shows the StructuredProductType complex type of the CCR schema modeled as an interconnection of UML classes. We represent each xs:complexType in the schema as a UML class with their respective UML stereotype. If an xs:element is a descendant of another schema concept, then this relation is represented as an equivalent class – subclass relation. This holds true for xs:sequence, xs:simpleType, etc. XML schema extensions (xs:extension) are represented as associations between classes. Data-type cardinality requirements (minOccurs, maxOccurs) and other XML constraints are represented with a «constraint» stereotype on the attribute. The xs:element type is represented with a «type» stereotype. Due to space limitations, we only show the Product xs:element and three main sub-elements: BrandName, ProductName, and Strength.

The next step is to apply security policies to the XSCD (top left of Fig. 1) by defining an XRSD that is capable of defining role-based access control policies or permissions on the attributes of the XSCD based on role, thereby achieving fine grained control. We note that permissions on XML documents are read, no read, write, and no write, represented in the XRSD as the respective stereotypes, «read/write», «read/nowrite», «noread/write», and «noread/nowrite». As an example, Fig. 3b defines Physician and Nurse XRSDs with permissions against the XSCD in Fig. 3a. Note that in Fig. 3b, the CCR complex type StructuredProductType element Product allows a role to have read and write permissions (Physician) or only read permissions (Nurse). While a Physician role can get all of the information regarding a drug and be able to create new instances following the schema, a Nurse role may be limited to read the drug details and cannot create new records.

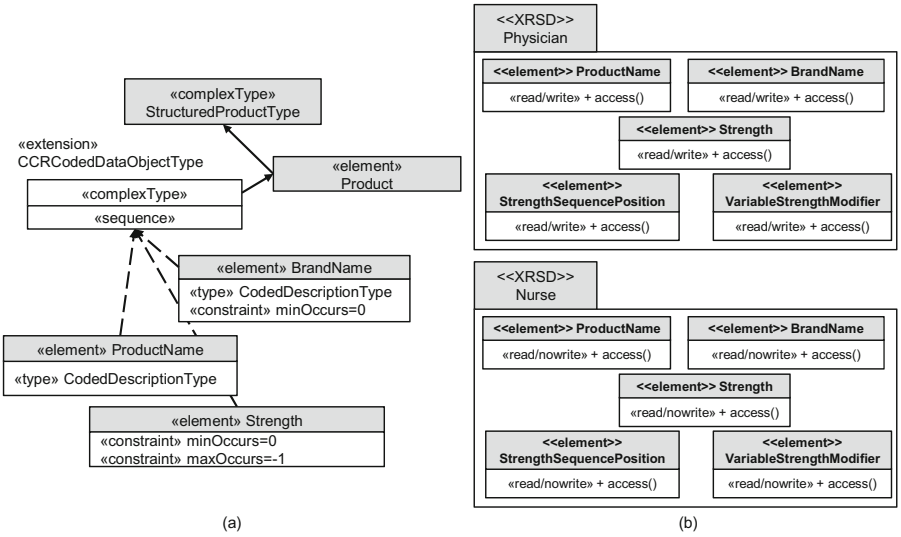

Fig. 3. XML Schema Class Diagram (a) and XML Role Slice Diagram (b).

4 Generating Policies from the XML Role Slice Diagram

In this section, we describe the generation of an XACML security policy (see Fig. 1 again) in order to allow XML instances to be customized and delivered to users based on role. As a result, security privileges defined at a schema level do not impact the XML instances of an application when privileges evolve, separating the security concern from the application data. By extracting the security policies targeting XML schemas and their instances into an external component of the framework, our approach avoids the high cost of updating XML schemas and instances when security policies change, in contrast to those approaches which embed the security policies as part of the XML schema and instance structure [5, 6].

To accomplish this, we present an approach to generating XACML security policies from the XRSD (see Fig. 3b). Towards this objective, Sect. 4.1 presents a process and architecture for the mapping of XRSDs that are used to generate a XACML policy for the schema based on the roles, using a portion of the CCR schema and its attributes; this achieves fine-grained control on CCR and results in an XACML policy that enforces the security as defined in XRSD against XSCD. Then, in Sect. 4.2, we present and explain an algorithm for this mapping process, which revolves around a set of equivalence rules between the XRSD and XACML structures; again, we use CCR as an example to illustrate the algorithm.

4.1 Mapping the XML Role Slice Diagram to the XACML Policy Construct

As given in Fig. 1, XRSDs (Fig. 3b) act as the blueprint of the role-based access-control policy for reading and writing permissions for a specific element or component of an XML schema for any given role, and represent the portions of the application's XML schemas that are to be allowed (or denied) access at an instance level. To map the XRSD into an XACML policy, we use XACML's language structure and processing model which consists of a PolicySet, a Policy, and a Rule. An XACML PolicySet is utilized to make the authorization decision via a set of rules in order to allow for access control decisions. A PolicySet can contain multiple Policy structures, and each Policy contains the access control rules. As a result, the Policy structure acts as the smallest entity that can be presented to the security system for evaluation. Based on our understanding of XACML and its usage, we are taking an approach that each XRSD must be mapped into a XACML Policy structure with its own set of rules that represent the appropriate enforcement for roles against a schema. Note that multiple XACML Policy structures may be generated, resulting in a PolicySet for a specific set of XML schemas that comprise a given application.

The collection of Policy structures is contained in a PolicySet, combined via an algorithm specified by the PolicySet's PolicyCombiningAlgId attribute that targets the particular XML schema. The XACML specification defines four standard combining algorithms: Deny-overrides (in which a policy is denied if at least one of the rules is denied); Permit-overrides (in which a policy is permitted if at least one of the rules is permitted); First-applicable (in which the result of the first rule's evaluation is treated as the result of all evaluations); and, Only-one-applicable (in which the combined result is the corresponding result to the acting rule). For our intent with XML instance security, and the way we map the XRSD into an XACML Policy, the combining algorithm of choice is Deny-overrides. With this algorithm, if a single Rule or Policy is evaluated to Deny, the evaluation result of the rest of the Rule elements under the policy is also Deny. While this might be the case when focusing on access control for XML instances in the document-level, as in our approach, other higher-level systems (e.g., software applications that utilize the XML instance, etc.) can very well deploy security policies with different combining algorithms.

In Fig. 5, we present the main sections of the mapped XACML policy for the Physician XRSD in Fig. 3b that is utilizing data as defined in the XSCD in Fig. 3a.

To create an XACML Policy structure per each XRSD, we present the following mapping equivalences and rules.

Policy and Rule Descriptors and Structure:

- Policy's PolicyId attribute value is the XRSD's Role value concatenated to AccessControlPolicy (e.g., the Physician role in Fig. 3b)
- Rule's RuleId attribute value is the XRSD's Role value concatenated to the XRSD's higher order element (e.g. in Fig. 3b it would be Product as defined in the XSCD in Fig. 3a), also concatenated to "ProductRule".
- Rule's Description value is the XRSD's Role concatenated to "Access Control Policy Rule".
- There are two XACML Rules under a higher level Target element, one for allowed and one for denied permissions.
- XACML Policy and Rules target and match the role (Subject, e.g., Physician in Figs. 3b and 4), the schema elements (Resources, e.g., ProductName, BrandName and Strength in Figs. 3a, b and 4), and the permissions (Actions, e.g., read and write in Figs. 3b and 4).

Rule Target's Subject (Fig. 4a):

- Only one XACML Subject and SubjectMatch per Rule.
- SubjectMatch's MatchId uses the function "string-equal" to evaluate the user's role as modeled in the XRSD.
- AttributeValue of the Subject is a string, and the value is the XRSD's Role (e.g., Physician in Figs. 3b and 4).
- SubjectAttributeDesignator's AttributeId is the role attribute.
- While more than one Rule per Policy might exist, the Subject is equal in both cases. This means that the role to be considered for policy evaluation is the same for operations that are allowed or denied.

Rule Target's Resources (Fig. 4b):

- One XACML Resource per permitted XRSD element.
- Each Resource's ResourceMatch has a MatchId that determines the usage of the function "string-equal".
- Resource's AttributeValue's is the XRSD's element names from the XSCD (e.g., ProductName, BrandName and Strength in Figs. 3a, b and 4).
- Resource's ResourceAttributeDesignator is an AttributeId that determines the target-namespace and datatype of the element.

Rule Target's Actions (Fig. 4c):

- One XACML Action per operation permitted exists (e.g., read and write in Figs. 3b and 4).
- ActionMatch's MatchId uses the function "string-equal".
- ActionAttributeDesignator's AttributeId value is action-write or action-read.

```
<Subjects>
    <Subject>
    <SubjectMatch MatchId="…:function:string-equal">
    <AttributeValue
     DataType="http://www.w3.org/2001/XMLSchema#string">
          Physician
    </AttributeValue>
    <SubjectAttributeDesignator AttributeId="…:attribute:role"
     DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
    </Subject>
</Subjects>
```

(a)

```
<Resources>
    <Resource>
    <ResourceMatch MatchId="…:function:string-equal">
    <AttributeValue DataType=" XMLSchema#string">
        ccr:schema:product:productname
    </AttributeValue>
    <ResourceAttributeDesignator
        AttributeId="…:resource:target-namespace"
        DataType=" XMLSchema#string"/>
    </ResourceMatch>
    </Resource>
    <Resource>
    <ResourceMatch MatchId="…:function:string-equal">
    <AttributeValue DataType="XMLSchema#string">
        ccr:schema:product:brandname
    </AttributeValue>
    <ResourceAttributeDesignator
        AttributeId="…:resource:target-namespace"
        DataType=" XMLSchema#string"/>
    </ResourceMatch>
    </Resource>
    <Resource>
    <ResourceMatch MatchId="…:function:string-equal">
    <AttributeValue DataType=" XMLSchema#string">
        ccr:schema:product:strength
    </AttributeValue>
    <ResourceAttributeDesignator
        AttributeId="…:resource:target-namespace"
        DataType=" XMLSchema#string"/>
    </ResourceMatch>
    </Resource>
</Resources>
```

(b)

```
<Actions>
    <Action>
    <ActionMatch MatchId="…:function:string-equal">
    <AttributeValue DataType="XMLSchema#string">
        read
    </AttributeValue>
    <ActionAttributeDesignator
        AttributeId="…:action:action-read"
        DataType="XMLSchema#string"/>
    </ActionMatch>
    </Action>
    <Action>
    <ActionMatch MatchId="…:function:string-equal">
    <AttributeValue DataType="XMLSchema#string">
        write
    </AttributeValue>
    <ActionAttributeDesignator
        AttributeId="…:action:action-write"
        DataType="XMLSchema#string"/>
    </ActionMatch>
    </Action>
</Actions>
```

(c)

Fig. 4. Mapped XACML Policy for the Physician role from the XRSD.

– ActionMatch's Attributevalue is the permission, read or write, depending on the stereotypes of the XRSD (e.g., read and write in Figs. 3b and 4).

Collectively, our approach presents three types of mappings: a *role mapping* (Fig. 4a) which maps a specific role (e.g., Physician) to a Policy's Subject; an *element mapping* (Fig. 4b),which maps an attribute (e.g., ProductName, Brand, Strength) to a Policy's Resource; and a *permission mapping* (Fig. 4c) which establishes permissions for the element (read and/or write) as Policy Actions. These mapping equivalences and rules permit each XACML Policy to capture the information modeled on the XRSD, while simultaneously limiting the amount of policies needed to only one per role. While each policy will have two high level Target elements, each with its own rules for those permissions that are allowed, the Effect of the Rule will be Permit, while those that are denied will have an Effect of Deny. Note that a special case is given to those roles where the permissions are all positive (a «read/write» stereotype in the XRSD) or all negative (a «noread/nowrite» stereotype in the XRSD). In these

cases, only one higher-level Target element with one Rule is necessary, and the positivity or negativity of the stereotype determines the Effect of the rule (if «read/write», then Permit, else if «noread/nowrite», then Deny).

4.2 Algorithm for the Mapping Process

The process of mapping the XRSDs to an XACML Policy can be automated, as shown by Fig. 5. The XRSD and schema to be secured serve as the parameters, while the XACML schema is used as template for the resulting instances. The first step of the algorithm determines whether or not all of the permission stereotypes in the XRSD are all positive or negative (either «read/write» or «noread/nowrite», respectively). If they are, then only one Target and Rule is needed to completely generate an equivalent Policy, and the algorithm proceeds down the left side branch. In this case, the algorithm proceeds through a series of steps. First, the template of the XACML Policy is created (based on the XACML schema) with one high-level target and rule. Depending on the permission stereotypes from the XRSD, the Policy Rule is set with an effect of Permit («read/write») or Deny («noread/nowrite»). Then, as shown in

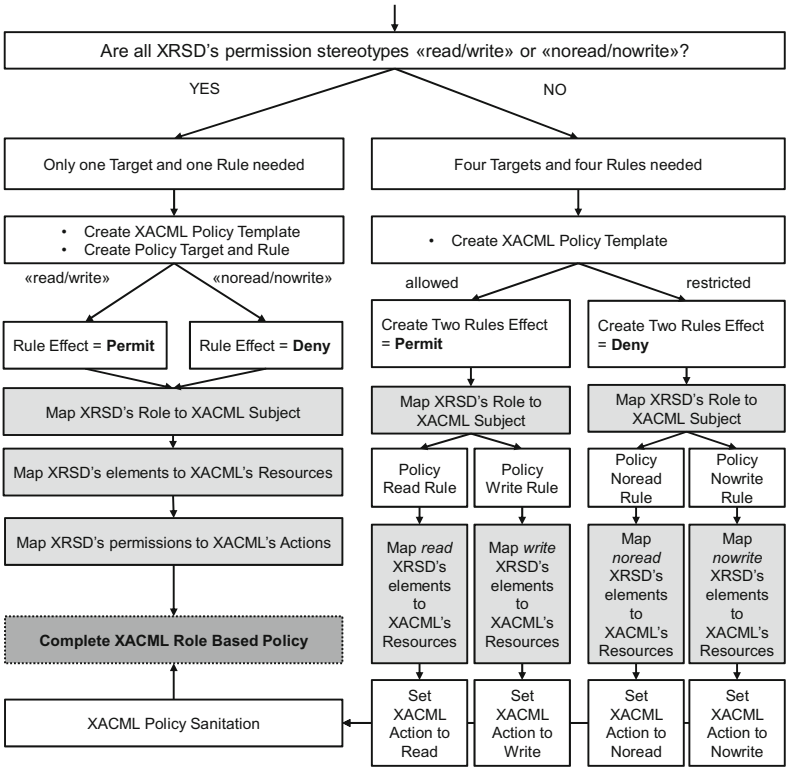

Fig. 5. Mapping from the XML Role Slice Diagram to the XACML Policy.

Fig. 4, a threefold mapping is performed between: the XRSD role and Rule's Subject; the XRSD elements and the Rule's Resources; and, the XRSD permission stereotypes and Rule's Actions; this finalizes the XACML Policy.

However, if not all permission stereotypes in the XRSD are all positive or negative, then the XACML Policy will require multiple high-level targets and rules, and the algorithm would proceed down the right side branch in Fig. 5. In this case, the first step is also creating the template XACML Policy, but with four high level Targets and Rules (two with the Effect of Permit, the others with the Effect of Deny). The fulfillment of these rules then depends on the permission stereotypes on each element. For those who have a positive permission (read or write), the elements are mapped as resources of the respective rule, and the permissions are mapped as actions. After the mapping process completes for each rule, the XACML Policy is finalized.

The enforcement process is straightforward. If a user has a role that has a no read permission (like the Nurse role), the policy enforcement point (PEP - or equivalent structure in the enforcing security architecture) filters the secured XML schema and the instance requested based on the permitted and allowed elements. For write operations, a similar enforcement takes place. These policies can also be applied to XSLT [4] or other query tools (e.g., XPath, XQuery, etc.) in order to provide filtered results to different role queries. This is an alternative to the traditional XML security approach of query rewrites, provided that the XSLT, XPath, and XQuery tools have a PEP that evaluates the XACML schema.

To summarize, Fig. 4 has the XACML policy created from the XRSD presented in Fig. 3b for the Physician role targeting the XML schema's Product element (note that because of space, not all equivalent XACML resources were included). The Physician role exhibits the special case of having all permissions allowed («read/write» on all elements). Because of this, only one Target with one Rule (with the Effect value of Permit) is needed. The Subject's AttributeValue is Physician (the role from the XRSD), and the resources are elements from the CCR schema (as also shown by the XRSD in Fig. 3b). Since the Physician role has both read and write permissions allowed for these elements, the two actions are part of the single Rule.

5 Policy Enforcement Process with Personal Health Assistant

In this section, we present the prototyping of the generated XACML policies on XML instances, transitioning from the mapping process is Sect. 4 to the enforcement process on the Personal Health Assistant (PHA) mobile application for health information management. In detail, in Sect. 5.1, we briefly review the general architecture for enforcement and its components (PHA, Microsoft HealthVault – MSHV - and our enforcement Middle-Layer Server). Section 5.2 presents the workflow utilized by the middle-layer server to enforce the permissions (read and write) set by the patient on the resulting XML instances.

5.1 General Architecture and Components

Personal Health Assistant (PHA) (Fig. 6) is a test-bed mobile application, developed in the University of Connecticut, for health information management that allows: patients to view and update their personal health record stored in their MSHV account and authorize medical providers to access certain portion of the protected health information (lower left); and, providers to obtain the permitted information from their respective patients (lower right). The patient version of PHA allows users to perform a set of actions regarding their health information (view and edit their medication list, allergies, etc.). Security settings can be set at a fine granular level, and using this information, policies are generated and stored in the patient's MSHV account (upper middle). The provider version of PHA allows the users (e.g., medical providers) to view and edit the medical information of their patients as long as they are permitted to do so as dictated by the security settings created by the patient.

In the overall architecture in Fig. 6, MSHV acts as the main data source and stores data in a proprietary structure that can be exported as XML structures, which in turn can be converted into a CCR compliant instance. To recreate the non-normative XACML architecture, our Middle-Layer Server acts as the policy access, information,

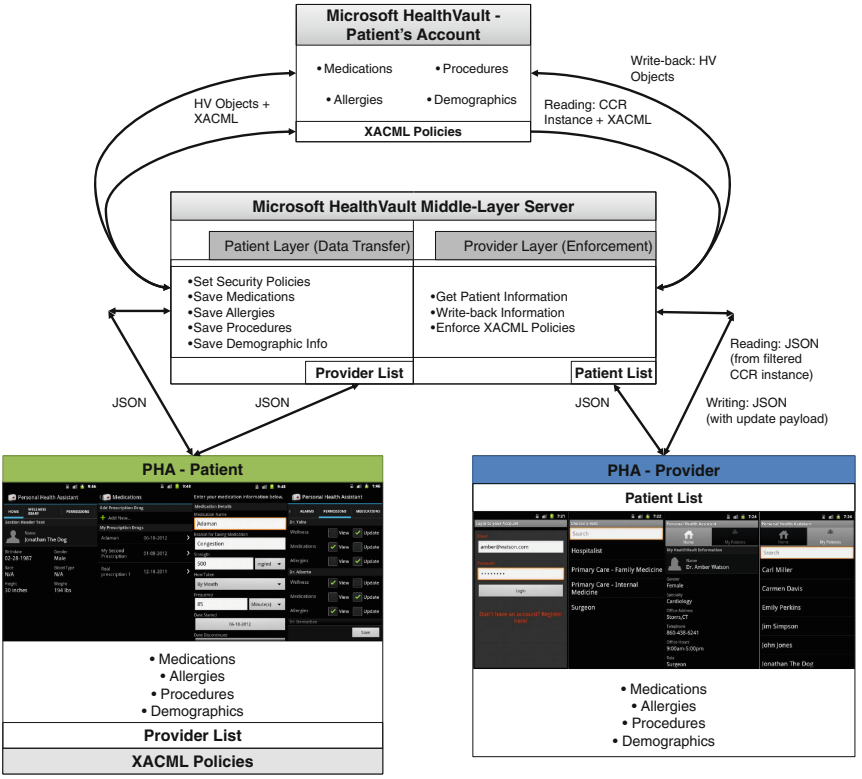

Fig. 6. PHA mobile applications and architecture.

decision, and enforcement points. To accomplish proper enforcement, we restrict all communication via our in-house developed Middle-Layer Server. With regards to data exchange, we have utilized JSON structures due to our familiarity and extensive experience with the format. Note that while we utilize JSON for transfers between PHA and the Middle-Layer Server, the security enforcement (done between the middle-layer server and MSHV) is performed on XML instances with XACML policies.

5.2 Enforcing XACML Policies on XML Instance and Segments

In this section, we describe the way that the XACML policy is enforced when handling reading and writing requests on XML instances whose schema has been secured when using the provider version of PHA. These two processes, though they utilize the same XACML policies to function, follow different workflows. We discuss the way that a medication object (StructuredProductType) from the CCR compliant instance from MSHV is secured (filtered) based on role and presented to the provider. We then explain the way that writing control is enforced with the same XACML policy.

The general process of securing the CCR instance for reading begins with a request from the provider version of PHA. When an initial request is made, the server retrieves the list of patients tied to the provider pertaining information. When a patient is selected, the server retrieves the corresponding XACML policy that targets the patient's information based on the requester's role. When a provider selects a category of health information (e.g., medications, procedures, etc.), the Middle-Layer Server, enforces the pertinent rules of the retrieved XACML policy. The process of this enforcement, as shown in Fig. 7a, involves the verification of the relevant rule (by evaluating the string representation of the users' role with the Subject role of the policy). After the relevant rule has been found (by utilizing the Resources' attributes), the reading permission is enforced by verifying it against the policy's Action elements.

If the action of the rule that is evaluated to Permit contains the read permission, then the CCR instance is not filtered. To support granular access control, recall from Sect. 4.2 that when stereotypes are not all-positive or all-negative (that is, there exists a combination of permissions over elements of the XML schema), more than one policy would match with respect to the role and resources. In this case, all policies will be evaluated and combined using the policy combination algorithm explained in Sect. 4.1. Once the CCR instance and segments have been filtered by the enforcement of the XACML policy, the resulting XML document is translated to JSON for the consumption of the provider version of the PHA application.

The process of securing the CCR schema for writing begins with a request from the provider's PHA. When a provider wants to update a patient's record (e.g., medication's StructuredProductType), the request is sent to the Middle-Layer Server tied to the update data as a JSON object, which verifies the target on which the rules of the requester's XACML Policy act upon. The server then evaluates the requester's role against the policy in order to determine if the write is allowed.

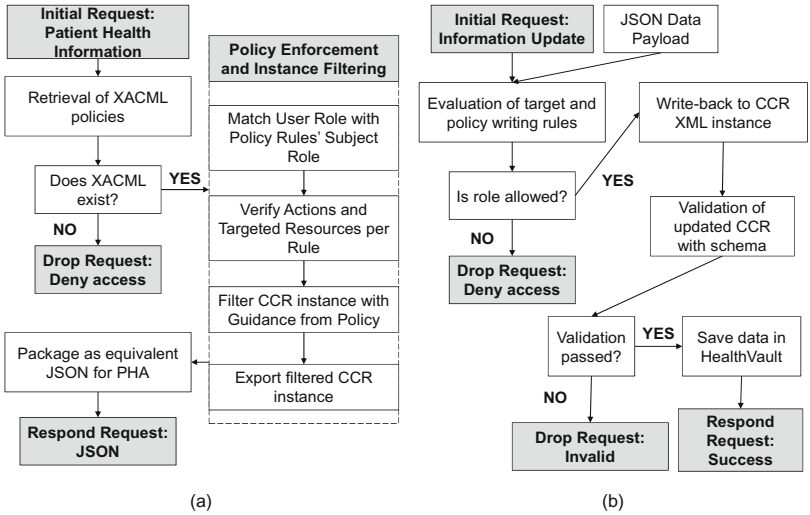

Fig. 7. Enforcing reading (a) and writing permissions (b) permissions in PHA.

The low-level enforcement of the XACML policy for writing permissions as given in Fig. 7b involves the same steps as when enforcing for reading (filtering) the document. If the user requesting an update operation has a role with a permission that allows it to occur (the write Action in the XACML Policy's Rule), the CCR instance is updated with the sent data, and validated with the CCR schema before the write-back to MSHV. If validation against the schema is successful, then the write-back occurs, and the update performed by the provider is saved in the patient's MSHV record. If the requester has a role that is not allowed to perform writing operations on the desired element, the request is dropped.

6 Conclusions and Ongoing Work

XML plays a pivotal role in the healthcare domain via the creation of standards such as CDA and CCR, which presents challenges in providing a robust security model for XML to ensure HIPAA compliance in the usage, transmission, and sharing of protected health information. To address this problem, our prior work [9] presented a security framework for XML that created UML-like artifacts for XML schemas and security, the XSCD and the XRSD. Using these as a basis, this paper has focused on the automatic generation of XACML policies from XRSDs (Sect. 4) that enforce the security defined on XML schemas against their corresponding instances. This allows the "same" instance to appear differently to specific users at a particular time. To demonstrate the feasibility and validity of our approach, Sect. 5 applied the generated XACML policies to the personal health assistant mobile application that allows patients to grant privileges to medical providers, and providers to view and update the data.

Our on-going work includes the extension of the work in this paper to support discretionary and mandatory access control, as well as applying our security framework to other platforms (e.g. Open *m*Health [11], etc.). These new approaches present many challenges; such as varied data representations (JSON, RDF, etc.), as well as the creation of more complex applications from the combination of different independent systems [7].

References

1. Baumer, D., Earp, J.B., Payton, F.C.: Privacy of medical records: IT implications of HIPAA. In: Tavani, H.T. (ed.) Ethics, Computing, and Genomics, pp. 137–152. Jones and Bartlett, Sudbury (2006)
2. Bertino, E., Carminati, B., Ferrari, E.: Access control for XML documents and data. Inf. Secur. Techn. Rep. **9**, 19–34 (2004)
3. Bertino, E., Ferrari, E.: Secure and selective dissemination of XML documents. ACM Trans. Inf. Syst. Secur. (TISSEC) **5**, 290–331 (2002)
4. Clark, J.: Xsl Transformations (Xslt). World Wide Web Consortium (W3C). http://www.w3.org/TR/xslt (1999)
5. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., et al.: Design and implementation of an access control processor for XML documents. Comput. Netw. **33**, 59–75 (2000)
6. Damiani, E., Fansi, M., Gabillon, A., et al.: A general approach to securely querying XML. Comput. Stan. Interfaces **30**, 379–389 (2008)
7. De la Rosa Algarín, A., Demurjian, S.A.: An approach to facilitate security assurance for information sharing and exchange in big data applications. In: Akhgar, B., Arabnia, H.R. (eds.) Accepted in Emerging Trends in Information and Communication Technologies Security. Elsevier, Amsterdam (2013)
8. De la Rosa Algarín, A., Demurjian, S.A., Ziminski, T.B., et al.: Securing XML with role-based access control: case study in health care. In: Ruiz Martínez, A., Pereñíguez García, F., Marín López, R. (eds.) Architectures and Protocols for Secure Information Technology, pp. 334–365. IGI Global, Hershey (2013)
9. De la Rosa Algarín, A., Demurjian, S. A., Berhe, S., et al.: A Security Framework for XML Schemas and Documents for Healthcare, pp. 782–789 (2012)
10. Dolin, R.H., Alschuler, L., Boyer, S., et al.: HL7 clinical document architecture, release 2. J. Am. Med. Inform. Assoc. **13**, 30–39 (2006)
11. Estrin, D., Sim, I.: Open mHealth architecture: an engine for health care innovation. Science **330**, 759–760 (2010). (Washington)
12. Ferraiolo, D.F., Sandhu, R., Gavrila, S., et al.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. (TISSEC) **4**, 224–274 (2001)
13. Kuper, G., Massacci, F., Rassadko, N.: Generalized XML security views. In: SACMAT 2005: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, pp. 77–84. ACM Press, New York (2005)
14. Leonardi, E., Bhowmick, S., Iwaihara, M.: Efficient database-driven evaluation of security clearance for federated access control of dynamic XML documents. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5981, pp. 299–306. Springer, Heidelberg (2010)

15. Müldner, T., Leighton, G., Miziołek, J.K.: Parameterized role-based access control policies for XML documents. Inf. Secur. J. A Globa. Persp. **18**, 282–296 (2009)
16. Pavlich-Mariscal, J.A., Michel, L., Demurjian, S.A.: A formal enforcement framework for role-based access control using aspect-oriented programming. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 537–552. Springer, Heidelberg (2005)

http://www.springer.com/978-3-662-44299-9

Web Information Systems and Technologies
9th International Conference, WEBIST 2013, Aachen, Germany, May 8-10, 2013, Revised Selected Papers
Krempels, K.-H.; Stocker, A. (Eds.)
2014, XII, 263 p. 88 illus., Softcover
ISBN: 978-3-662-44299-9