Defining and Enforcing XACML Role-based Security Policies within an XML Security Framework

Alberto De la Rosa Algarín[1], Timoteus B. Ziminski[1], Steven A. Demurjian[1], Robert Kuykendall[2] and Yaira K. Rivera Sánchez[1]
[1]*Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, U.S.A.*
[2]*Department of Computer Science, Texas State University, San Marcos, TX, U.S.A.*

Keywords: Security and Policy Modeling, Security Policies, XML, XACML, Role-based Access Control.

Abstract: Securing electronic data has evolved into an important requirement in domains such as health care informatics, with the eXtensible Markup Language (XML) utilized to create standards such as the Clinical Document Architecture and the Continuity of Care Record, which have led to a need for approaches to secure XML schemas and documents. In this paper, we present a method for generating eXtensible Access Control Markup Language (XACML) policies that target XML schemas and their instances, allowing instances to be customized for users depending on their roles. To do so, we extend the Unified Modeling Language (UML) with two new diagrams to model XML: the XML Schema Class Diagram (XSCD) to define the structure of an XML document in UML style; and the XML Role-Slice Diagram (XRSD) to define roles and associated privileges at a granular access control level. In the process, we separate the XML schemas of an application from its security definition in XRSD. To demonstrate the enforcement of our approach, we utilize a personal health assistant mobile application for health information management, which allows patients to share personal health data with providers utilizing XACML for security definition.

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) (Dolin et al., 2006) for health information exchange, and the Continuity of Care Record (CCR) for capturing clinical patient data. In such settings, both security and privacy protection must be insured so individuals have 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) (Baumer et al., 2000), which provides a set of security guidelines in the usage, transmission, and sharing of protected health information. 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 XML instances to be filtered (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 secure information engineering paradigm, we have defined a security framework for XML in prior work (De la Rosa Algarín et al., 2012)

De la Rosa Algarín A., B. Ziminski T., A. Demurjian S., Kuykendall R. and K. Rivera Sánchez Y..
Defining and Enforcing XACML Role-based Security Policies within an XML Security Framework.
DOI: 10.5220/0004366200160025
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 16-25
ISBN: 978-989-8565-54-9

as shown in Figure 1. The general approach is to have a set of XML schemas corresponding to an application (middle right in Figure 1), which will be instantiated for the executing application (bottom right of Figure 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 Figure 1). To achieve this in a secure information engineering context, the framework in Figure 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.

The purpose of this paper is to extend our earlier work (De la Rosa Algarín et al., 2012) by concentrating on the left hand side of Figure 1 (the XACML Policy Mapping box) to focus on 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 Section 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 Section 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 Section 5, we demonstrate the XACML policy interplay and enforcement with PHA, describing 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 concludign remarks and ongoing work in Section 6.

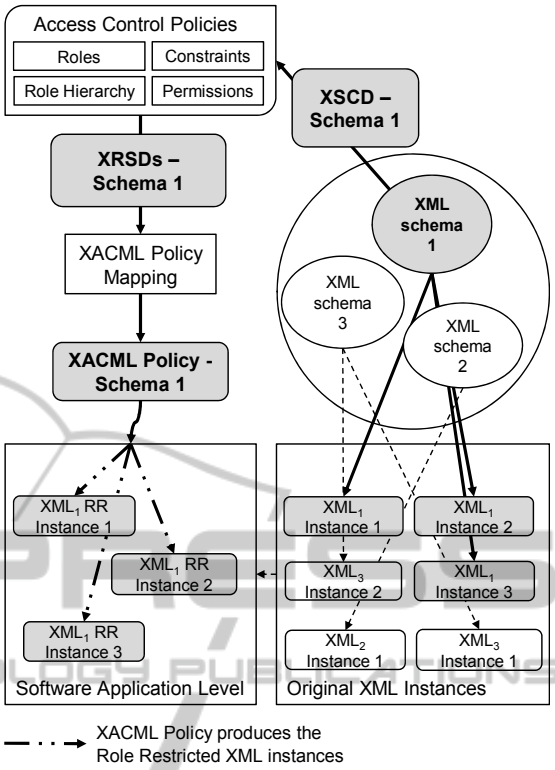

Figure 1: Security Framework for XML.

2 RELATED WORK

The work of (Damiani et al., 2000) presents an access control system that embeds the definition and enforcement of the security policies in the structure of the XML documents in order to provide customizable security. The security details can be embedded in the XML DTD, providing a level of generalization for documents that share the same DTD. This is similar to our work in that security policies act in both a descriptive level of the XML instances and target the XML instances. However, there are two differences. First, their work targets XML DTD's (outdated XML), while ours utilizes schemas. Second, their security policies are embedded into both the DTD and the instance, requiring changes to instances; our work allows policy changes with no impact on instances.

Another effort (Damiani et al., 2008) 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 in that it targets security in XML document instances via

policies, it differs by requiring changes to instance when the policy is modified and does not consider XML document writing as we do (see Section 5.3).

Efforts by (Bertino and Ferrari, 2002); (Bertino et al., 2004) 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 base 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 (Leonardi et al., 2010) 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 (Kuper et al., 2005) has presented a model consisting of access control policies over DTD's (again, outmoded in XML) with XPath expressions in order to achieve XML security. The purpose of 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 (Müldner et al., 2009) 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. This approach supports the encryption of segments of the XML document. 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 (Ferraiolo et al., 2001) standard is an access control model where permissions are assigned to roles, and roles are assigned to users. NIST RBAC has four reference models ($RBAC_0$, $RBAC_1$, $RBAC_2$ and $RBAC_3$). 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 separation 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. Information can be 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 CCR standard allows the creation of documents that include patient information (demographics, social security number, insurance policy details, medications, procedures, etc.) with a common structure for a more uniform information exchange across institutions that require its usagea. The CCR schema contains elements for virtually all health information items, and is represented with extended granularity for better detail keeping. For example, and for reader understanding of the following sections, Figure 2 shows a subset of the official CCR schema. This fraction corresponds to the complexType element *StructuredProductType*, which is utilized to represente medications and all their attributes. This *StructuredProductType* is utilized 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 (Pavlich-Mariscal et al., 2008) via the UML meta-model. Using this as a basis, we have extended this work to define two new UML artifacts (De la Rosa Algarín et al., 2012): 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,

As an example, consider the XSCD shown in Figure 3, where the *StructuredProductType* complex type of the CCR schema is modelled 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. Note that due to space limitations, we only show the representation of the *Product* xs:element and three main sub-elements: *BrandName*, *ProductName*, and *Strength*.

```
<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>
```

Figure 2: Segment of the Continuity of Care Record Schema's StructuredProductType.

The next step is to apply security policies to the XSCD (top left of Figure 1) by defining a new UML-like diagram: the XML Role Slice Diagram, XRSD, that is capable of defining 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, Figure 4 defines Physician and Nurse XRSDs with permissions against the XSCD in Figure 3. Note that in Figure 4, 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.

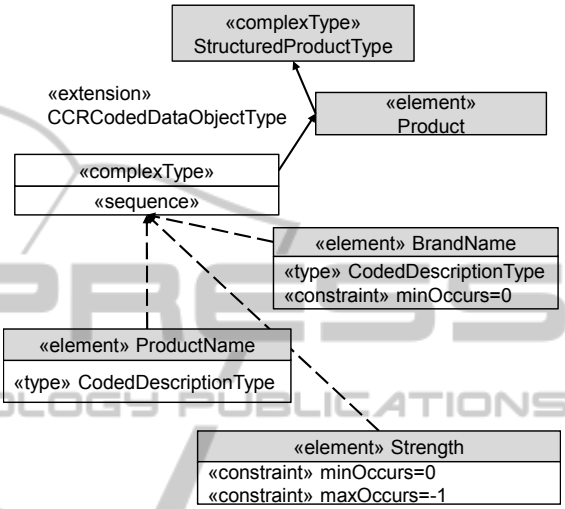

Figure 3: XSCD of the StructuredProductType.

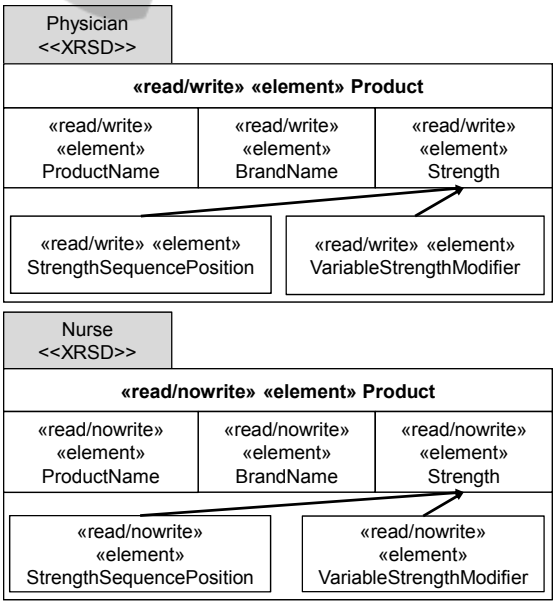

Figure 4: XRSD of a Health Care Scenario with the Product Element of StructuredProductType.

4 GENERATING POLICIES FROM THE XRSD

In this section, we describe the generation of an XACML security policy (see Figure 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 this we mean that, 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 (Damiani, 2000; Damiani, 2008).

To accomplish this, we present an approach to generating XACML security policies from the XRSD (see Figure 4). Towards this objective, Section 4.1 presents a process and architecture for the mapping of XRSDs that are used in conjunction 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 Section 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 utilize CCR as an example to illustrate the algorithm.

4.1 Mapping the XRSD to the XACML Policy Construct

As given in Figure 1, XRSDs (Figure 4) act as the blueprint of the access-control policy for reading and writing permissions for a specific element or component of an XML schema for any given role, and are used to represent the portions of the application's XML schemas (XCSD – Figure 3) that are to be allowed (or denied) access at an instance level. To map the XRSD into an XACML policy, we utilize the policies' language structure and processing model. XACML policies consist 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 Figure 5, we present the main sections of the mapped XACML policy for the Physician XRSD in Figure 4 that is utilizing data as defined in the XCSD in Figure 3. 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 Figure 4)
- Rule's *RuleId* attribute value is the XRSD's *Role* value concatenated to the XRSD's higher order element (e.g. in Figure 4 it would be *Product* as defined in the XCSD in Figure 3), 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 Figure 4 and 5), the schema elements (*Resources*, e.g., **ProductName**, **BrandName** and **Strength** in Figure 3, 4 and 5), and the permissions (*Actions*, e.g., **read** and **write** in Figure 4 and 5).

Rule Target's Subject (Figure 5a):

- 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 Figure 4 and 5).
- *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 (Figure 5b):

- 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* value is the XRSD's element names from the XCSD (e.g., **ProductName**, **BrandName** and **Strength** in Figure 3, 4 and 5), referencing the original schema.
- Resource's *ResourceAttributeDesignator* is an *AttributeId* that determines the target-namespace and datatype of the element.

Rule Target's Actions (Figure 5c):

- One XACML *Action* per operation permitted exists (e.g. **read** and **write** in Figure 5 and 6).
- *ActionMatch's MatchId* uses the function "string-equal".
- *ActionAttributeDesignator's AttributeId* value is *action-write* or *action-read*.
- *ActionMatch's Attributevalue* is the permission, *read* or *write*, depending on the stereotypes of the XRSD (e.g., **read** and **write** in Figure 4 and 5).

```
<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>                                              (a)
</Subjects>

<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>                                             (b)
</Resources>

<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>                                               (c)
</Actions>
```

Figure 5: Mapped XACML Policy for Physician Role from XRSD.

Collectively, our approach presents three types of mapping: a *role mapping* (Figure 5a), which maps a

specific role (e.g., Physician) to a Policy's Subject; an *element mapping* (Figure 5b), which maps an attribute (e.g., ProductName, Brand, Strength) to a Policy's Resource; and a *permission mapping* (Figure 5c), 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*), 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 XRSD to an XACML Policy can be automated, as shown by Figure 6. The XRSD and schema to be secured serve as the parameters, while the XACML schema is utilized as template for the resulting instances. The first step of the algorithm is determining whether or not all of the permission stereotypes in the XRSD are all positive or all negative (either «read/write» or «noread/nowrite», respectively). If they are, then we know that only one Target and Rule is needed to completely generate an equivalent Policy, and the algorithm proceeds down the right 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 Figure 5, 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.

Alternatively, if not all of the permission stereotypes in the XRSD are all positive or all negative, then the XACML Policy will require two high-level targets and rules, and the algorithm would proceed down the left side branch. In this case, the algorithm proceeds in a series of alternative steps. The first step is also creating the template XACML Policy, but with two high level Targets and two Rules (one with the Effect of Permit, the other with the Effect of Deny). The fulfilment of these two rules then depends on the permission stereotypes on each element. For those who have a positive permission (either read or write), the elements are mapped as resources of the respective rule; and the permissions are mapped as actions. Note that while two rules exist in this case, the subject will be the same on both (the XRSD role Physician). After this mapping process is complete for each rule, the XACML Policy is finalized.

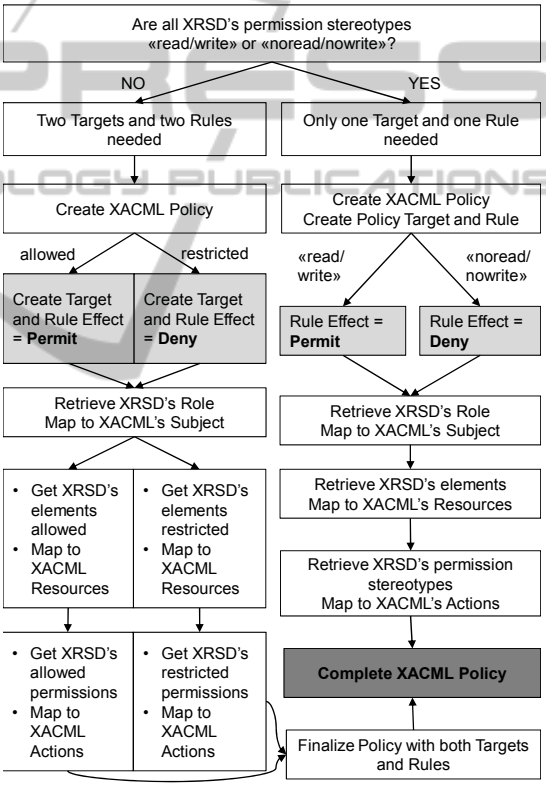

Figure 6: Mapping from XRSD to XACML Policy.

From an enforcement perspective, in support of either mapping, the process is relatively straightforward. If a user has a role that has a no read permission (like the Nurse role in Figure 4), the policy enforcement point (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 (Clark,

1999) or other query tools (e.g., XPath, XQuery, etc.) in order to provide filtered results to different role queries, an alternative to the more traditional XML security approach of query rewrites, provided that the XSLT, XPath, and XQuery tools have a policy enforcement point (PEP) that complies with the non-normative XACML architecture. In this approach, the result of the query will be the original XML instance, and the PEP will perform all of the filtering.

To summarize, Figure 5 shows the resulting XACML policy created from the XRSD presented in Figure 4 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 XRSD 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 Figure 4). 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 PHA

In this section, we present the prototyping efforts of enforcing the generated XACML policies on XML instances, transitioning from the mapping process is Section 4 to demonstrate the enforcement process on the Personal Health Assistant (PHA) mobile application for health information management. To accomplish this, in Section 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 CCR instances, focusing on the recurring example of medications and the CCR StructuredProductType.

5.1 General Architecture and Components

Personal Health Assistant (PHA) 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 (patient version); and, providers to obtain the permitted information from their respective patients (provider version). The patient version of PHA allows users to perform a set of actions regarding their health information (view and edit their medication list, allergies, procedures, etc.). Security settings can be set at a fine granular level, and each provider receives view/update authorizations to the different information components available in PHA on a patient-by-patient basis. Using this information, policies are generated and stored in the patient's MSHV account. 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, Microsoft HealthVault (MSHV) acts as the main data source. MSHV 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 MSHV Middle-Layer Server acts as the policy access, information, decision, and enforcement points. To accomplish proper enforcement, we restrict all communication to MSHV 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 Instances 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 how a medication object (StructuredProductType) from the CCR compliant instance from MSHV is secured (filtered) based on roles. Next, we explain how 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 Figure 7, 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 Section 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 Section 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 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.

The low-level enforcement of the XACML policy for writing permissions as given in Figure 8 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.

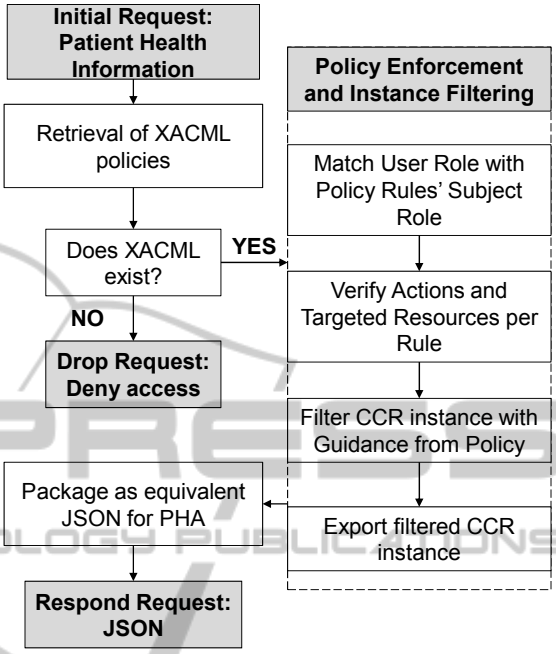

Figure 7: Enforcing Reading Permissions (Filtering) on XML Instances in PHA.

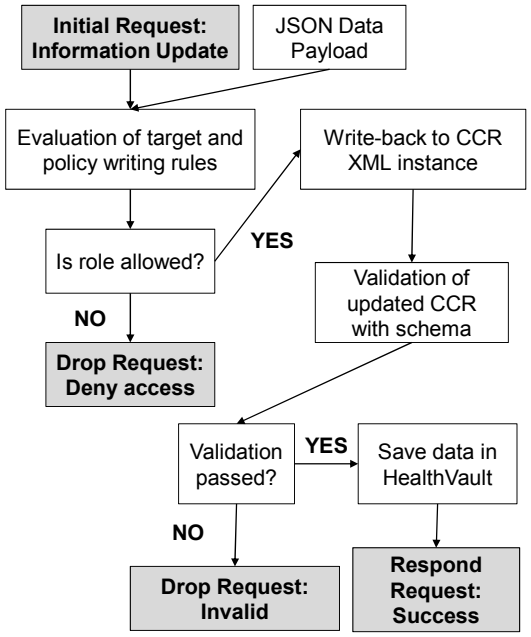

Figure 8: Enforcing Writing Permissions on XML Instances in PHA.

6 CONCLUSIONS AND ONGOING WORK

XML plays a pivotal role in the biomedical and healthcare domains via the creation of standards such as CDA and CCR. These domains present 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 (De la Rosa Algarín, 2012) 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 (Section 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, Section 5 applied the generated XACML policies to the PHA application for health information management that allows patients to grant privileges to medical providers, and providers to view and update the data. Our prototype, using Microsoft HealthVault as a backend with our own middle-layer server to enforce the generated XACML policies, provides an important proof of concept to the work presented herein.

Our on-going work is in a number of different areas. As XACML continues to evolve, with support for duty delegation and role delegation is being formalized in both the policy language and processing model; we are looking at how to model this with our established XSCD and XRSD artifacts. We are also looking at applying our security framework and the work presented in this paper to other health IT platforms, such as the SMART Platform (http://smartplatforms.org/), and Open mHealth (Estrin and Sim, 2010). These new approaches to healthcare informatics present many challenges, such as the use of different security policies based on the data source, and the various data structure utilized to represent information (e.g., JSON, RDF, OWL, etc.), as well as the creation of more complex systems and/or applications that result from the combination of different independent systems and/or applications.

REFERENCES

Baumer, D., Earp, J. and Payton, F. 2000. Privacy of medical records: IT implications of HIPAA. *ACM SIGCAS Computers and Society*, 30, 4, 40-47.

Bertino, E. and Ferrari, E. 2002. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 2002, 5, 290-331.

Bertino, E., Castano, S., Ferrari, E. and Mesiti, M. 2002. Protection and administration of XML data sources. *Data & Knowledge Engineering, Elsevier*, 2002, 43, 237-260.

Bertino, E., Carminati, B. and Ferrari, E. 2004. Access control for XML documents and data. *Information Security Technical Report, Elsevier*, 2004, 9, 19-34.

Clark, J. et al. 1999. XSL transformations (xslt) version 1.0. *W3C Recommendation*, 16, 11, 1999.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P., 2000. Design and implementation of an access control processor for xml documents. *Computer Networks*, 33, 1, 59–75.

Damiani, E., Fansi, M., Gabillon, A. and Marrara, S. 2008. A general approach to securely querying xml. *Computer Standards & Interfaces*, 30, 6, 379–389.

De la Rosa Algarín, A., Demurjian, S., Berhe, S., Pavlich-Mariscal, J. 2012. A Security Framework for XML schemas and Documents for Healthcare. *Proceedings of 2012 International Workshop on Biomedical and Health Informatics (BHI 2012)*, 782-789.

Dolin, R.H., Alschuler, L., Boyer, S., Beebe, C., Behlen, F.M., Biron, P.V. and Shvo, A.S. 2006. HL7 clinical document architecture, release 2. *Journal of the American Medical Informatics Association*, 13, 1, 30-39.

Estrin, D., and Sim, I. 2010. Open mHealth architecture: an engine for health care innovation. *Science* (Washington), 330 (6005), 759-760.

Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R. and Chandramouli, R. 2001. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4, 3, 224-274.

Kuper, G., Massacci, F. and Rassadko, N. 2005. Generalized XML security views. *Proceedings of the tenth ACM symposium on Access control models and technologies*, 2005, 77-84.

Leonardi, E., Bhowmick, S. and Iwaihara, M. 2010. Efficient database-driven evaluation of security clearance for federated access control of dynamic XML documents. *Database Systems for Advanced Applications*, 2010, 299-306.

Müldner, T., Leighton, G. and Miziołek, J. 2009. Parameterized Role-Based Access Control Policies for XML Documents. *Information Security Journal: A Global Perspective, Taylor & Francis*, 2009, 18, 282-296.

Pavlich-Mariscal, J., Demurjian, S. and Michel, L. 2008. A framework of composable access control definition, enforcement and assurance. *SCCC'08. International Conference of the IEEE*, 2008, 13–22.