XACML Use Case for XML Fine-grained Access Control

March 10, 2002

Author: Michiharu Kudo

This document describes an XACML use case for fine-grained XML resource protection. A couple of policy specification examples are specified using future XACML extension feature. The policy specification is based on the latest XACML language proposal document [1] and SAML specification [2].

1. Overview of XML Fine-grained Access Control

1.1 Data-flow

Figure 1 shows a data-flow diagram for XML fine-grained access control.

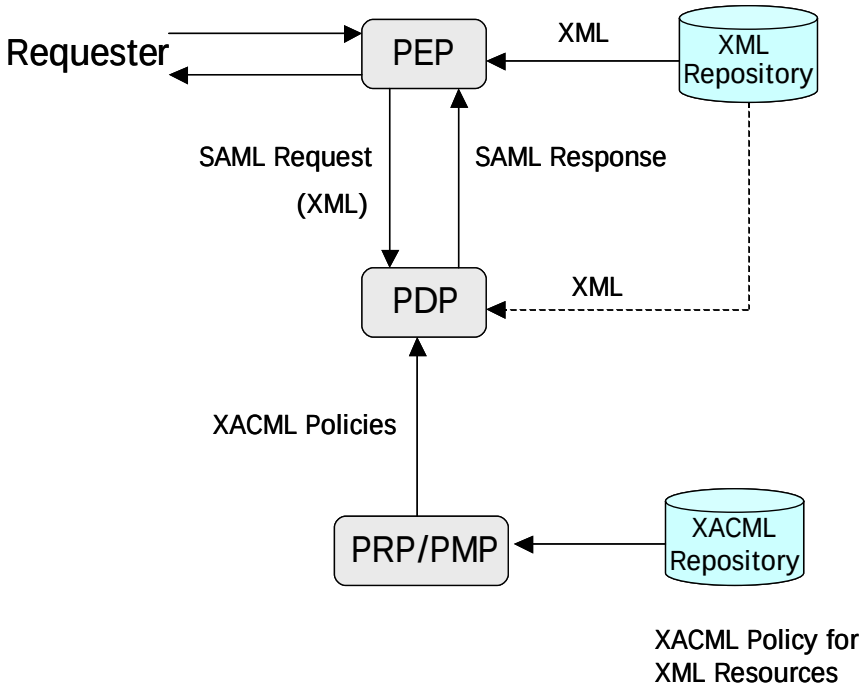

Figure 1 Data-flow diagram

Since target resource is XML, it is stored in XML repository (file system, etc.). The target XML may be sent to PDP by embedding in SAML request, accessed directly by PDP (or PIP), or may not be referred.

1.2 Language Primitives

Figure 2 shows fundamental XACML language components used for XML fine-grained access control.

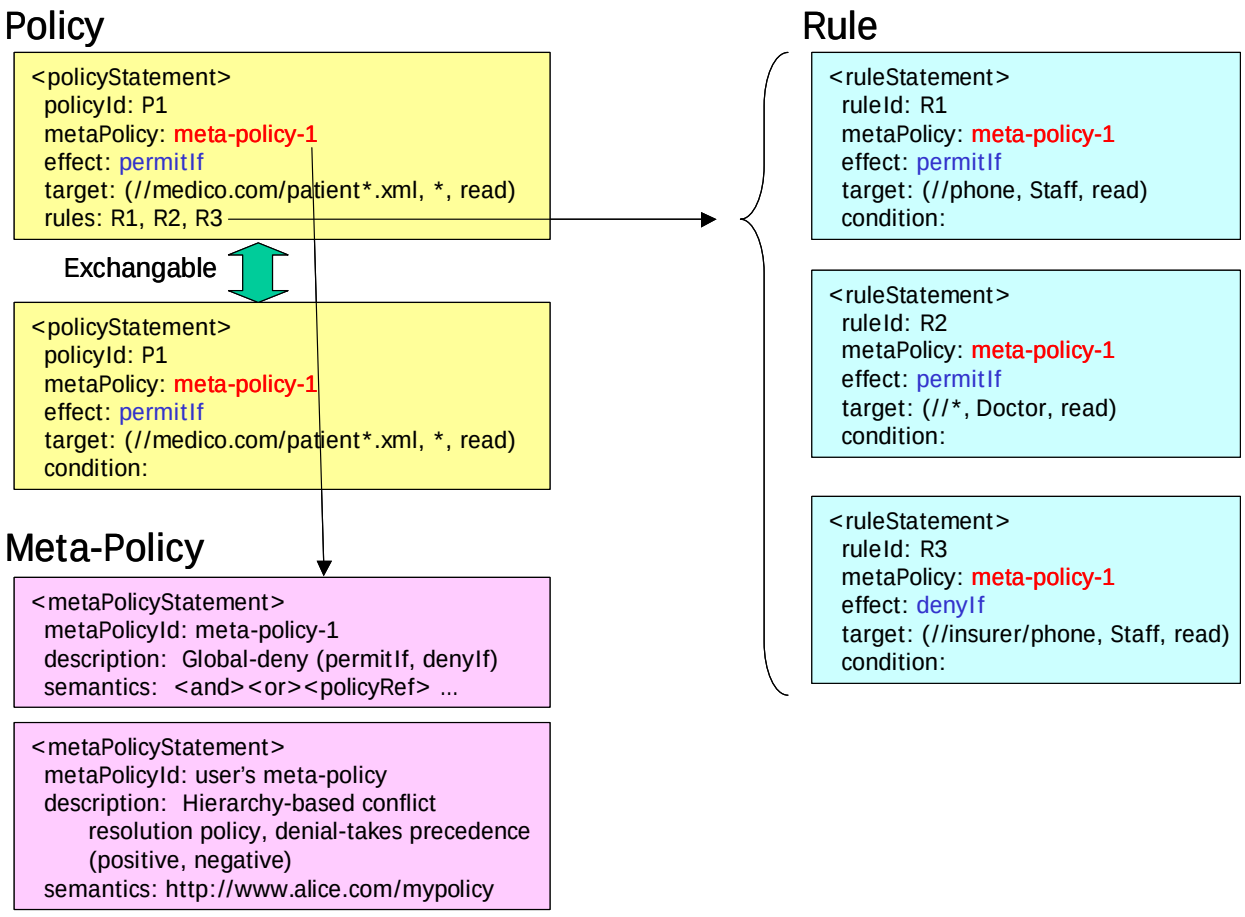

Figure 2. Fundamental XACML language components

2. XML Resource Reference

From XML resource protection viewpoint, it is important to define which granularity of the XML resource can be referred by the access control policy. We consider three kinds of reference methods:

1. XPath
2. Simple path expression
3. ID reference

XPath resource reference is that any resource units that can be specified using XPath are possible resource unit. Elements and attributes are typical reference unit. Simple path expression is similar to the usual path expression used in file systems (UNIX etc.). The difference between XPath and the simple path expression is briefly described in Section 2.5. ID reference is based on the data type-based reference method. Both DTD and XML Schema support this ID notion.

In this section, we only give a couple of authorization request examples of SAML Request / Response. Corresponding access control policy examples are described in the next section. We are assuming that the PEP sends a SAML authorization decision request to the PDP after the PEP identifies a target XML resource in response to the resource request from the request initiator. We do not consider cases such that multiple XML resources are referred in the SAML authorization decision request.

2.1 XPath Resource Reference

2.1.1 Access to a specific node of a specific XML file

This example shows a case that a requester needs to access a specific node of XML document. For example, a hospital staff may need to read a patient's date of birth stored in <patientDoB> element of the patient-123.xml. Target resource reference is encoded in the SAML AuthorizationDecisionQuery as follows:

```
<AuthorizationDecisionQuery Resource="http://medico.com/patient-123.xml#xpointer(/record/patient/patientDoB)">
…
</AuthorizationDecisionQuery>
```

Target resource reference uses an XPointer syntax [3][1]. The string before "#" character indicates URI [4] of the target XML resource and the string after "#xpointer(" indicates XPath expression [5]. In this example, the URI is "http://medico.com/patient-123.xml" and the XPath is "/record/patient/patientDoB". In this case, this XPath refers to only one node (element). Refer to Section 2.1.3 where XPath expression refers to a set of nodes.

In the above example, we assumed that each element of the target XML is uniquely identified without namespace identifier. When namespace identifier is necessary, the XPath expression looks like:

```
<AuthorizationDecisionQuery Resource="http://medico.com/patient-123.xml#xmlns (x=http://medico.com)
xpointer(/x:record/x:patient/x:patientDoB)">
…
</AuthorizationDecisionQuery>
```

[1] XACML uses a subset of XPointer specification that allows only XPath expression in "#xpointer() function. The status of the latest XPointer specification is a candidate recommendation.

The above specification means that each element in the patient-123.xml is accompanied with the namespace URI “http://medico.com”.

The URI argument can be an arbitrary URI syntax such as “file://c:/winnt/system32/wmpscheme.xml”. In this case, the PDP and the PEP must have the identical meaning about the location of the target resource.

```
<AuthorizationDecisionQuery Resource=” file://c:/winnt/system32/wmpscheme.xml#xpointer(/mediaindexscheme)”>
…
</AuthorizationDecisionQuery>
```

SAML response looks like:

```
<Response xmlns:=”…draft-sstc-schema-protocol-24.xsd” xmlns:saml=” …draft-sstc-schema-assertion-24.xsd” >
  <saml:Assertion>
    <saml:AuthorizationDecisionStatement
Resource=” ”http://medico.com/patient-123.xml#xpointer(/record/patient/patientDoB)” Decision=”Permit”>
      <saml:Subject>
        <saml:NameIdentifier SecurityDomain=”foo” Name=”baa”/>
      </saml:Subject>
      <saml:Actions>
        <saml:Action Namespace=”http://www.oasis-open.org/…/xmlactions”>Read</saml:Action>
      </saml:Actions>
    </saml:AuthorizationDecisionStatment>
  </saml:Assertion>
</Response>
```

2.1.2 Access to an XML document embedded in SAML request

This example shows the case where the target XML resource is embedded in the SAML request as well as the SAML authorization decision query.

```
<AuthorizationDecisionQuery Resource=”xpointer(/record/patient/patientDoB)”>
…
</AuthorizationDecisionQuery>
```

The URI description is not in the Resource attribute. This means that the target XML is embedded in the SAML request. The example below shows how the SAML authorization decision request carries the target XML document. The target XML is indicated in blue color that is inserted in the ResourceStatement element under the AuthorizationDecisionQuery/Evidence/Assertion element in the SAML request.

```
<Request xmlns:=”…draft-sstc-schema-protocol-24.xsd” xmlns:saml=” …draft-sstc-schema-assertion-24.xsd” >
  <AuthorizationDecisionQuery                                    Resource=”xmlns(md=medico.com/records.xsd)
xpointer(/md:record/md:patient/md:patientDoB)”>
    <saml:Subject>
      <saml:NameIdentifier SecurityDomain=”foo” Name=”baa”/>
    </saml:Subject>
    <saml:Actions>
      <saml:Action Namespace=”http://www.oasis-open.org/…/xmlactions”>Read</saml:Action>
    </saml:Actions>
    <saml:Evidence>
      <saml:Assertion AssertionID=”12345” …>
        <samle:ResourceStatement xmlns:samle=”…samlExt.xsd”>

          <md:record xmlns:md=”medico.com/records.xsd”>
            <md:patientName>
              <md:first>Bartholomew</md:first>
```

```
                    <md:last>Simpson</md:last>
                  </md:patientName>
                  …
                </md:record>

            </samle:ResourceStatement>
          </saml:Assertion>
        </saml:Evidence>
      </AuthorizationDecisionQuery>
    </Request>
```

The PDP deals with the XPath of the target resource reference as if the root element of the target resource were located just below the samle:ResourceStatement element. The samle:ResourceStatement element is an extension point that allows any schema below it.

2.1.3 Access to a set of nodes in an XML document

There are some cases where the XPath expression of the target resource reference refers to more than one node. We allow this flexibility by returning multiple access decision assertions in response to the set of nodes requested.

```
  <AuthorizationDecisionQuery Resource="http://medico.com/patient-123.xml#xpointer(//email)">
  …
  </AuthorizationDecisionQuery>
```

The above query is asking access decisions for every email elements in the patient-123.xml. Since there are four email elements, the resultant SAML response includes four authorization decision statements (we omitted namespace prefix "md:" in the Resource attribute for brevity):

```
  <Response xmlns:="…draft-sstc-schema-protocol-24.xsd" xmlns:saml=" …draft-sstc-schema-assertion-24.xsd" >
    <saml:Assertion>
      <saml:AuthorizationDecisionStatement Resource="xpointer(/record/patient/patientContact/email)"
Decision="Permit">
        <saml:Subject>
          <saml:NameIdentifier SecurityDomain="foo" Name="baa"/>
        </saml:Subject>
        <saml:Actions>
          <saml:Action Namespace="http://www.oasis-open.org/…/xmlactions">Read</saml:Action>
        </saml:Actions>
      </saml:AuthorizationDecisionStatment

      <saml:AuthorizationDecisionStatement Resource="
xpointer(/record/patientGuardian/patientGuardianContact/email)" Decision="Permit">
        <saml:Subject>
          <saml:NameIdentifier SecurityDomain="foo" Name="baa"/>
        </saml:Subject>
        <saml:Actions>
          <saml:Action Namespace="http://www.oasis-open.org/…/xmlactions">Read</saml:Action>
        </saml:Actions>
      </saml:AuthorizationDecisionStatment

      <saml:AuthorizationDecisionStatement Resource="
xpointer(/record/primaryCarePhysician/physicianContact/email)" Decision="Deny">
        <saml:Subject>
          <saml:NameIdentifier SecurityDomain="foo" Name="baa"/>
        </saml:Subject>
        <saml:Actions>
          <saml:Action Namespace="http://www.oasis-open.org/…/xmlactions">Read</saml:Action>
        </saml:Actions>
      </saml:AuthorizationDecisionStatment
```

```
        <saml:AuthorizationDecisionStatement Resource="xpointer(/record/insurer/email)" Decision="Deny">
          <saml:Subject>
            <saml:NameIdentifier SecurityDomain="foo" Name="baa"/>
          </saml:Subject>
          <saml:Actions>
            <saml:Action Namespace="http://www.oasis-open.org/…/xmlactions">Read</saml:Action>
          </saml:Actions>
        </saml:AuthorizationDecisionStatment
      </Assertion>
    </Request>
```

In similar way, it is possible to submit a query about every node included in a specific sub-tree. The query in the next example shows an access request to every node in the medical information in the medical-123.xml.

```
    <AuthorizationDecisionQuery Resource="http://medico.com/patient-123.xml#xpointer(/record/medical//*)">
    …
    </AuthorizationDecisionQuery>
```

2.1.4 Access to a set of XML document

There are some cases where the target XML resource is not only one XML document. We allow this flexibility.

```
    <AuthorizationDecisionQuery
Resource="http://medico.com/records#xpointer(/record/patient[@patientName/first='Alice']/patientDoB)">
    …
    </AuthorizationDecisionQuery>
```

Above query assumes that the resource URI "http://medico.com/records" means a collection of XML documents which schema is "medico.com/records.xsd". If the resource URI contains more than one document, the PDP and the PEP must share the notation to indicate the unique identifier of the document. For example, if there are two XML documents, ID1212 and ID1212, then the SAML response looks like:

```
    <Response xmlns:="…draft-sstc-schema-protocol-24.xsd" xmlns:saml=" …draft-sstc-schema-assertion-24.xsd" >
      <saml:Assertion>
        <saml:AuthorizationDecisionStatement
Resource=" "http://medico.com/records/ID1212#xpointer(/record/patient/patientDoB)" Decision="Permit">
          <saml:Subject>
            <saml:NameIdentifier SecurityDomain="foo" Name="baa"/>
          </saml:Subject>
          <saml:Actions>
            <saml:Action Namespace="http://www.oasis-open.org/…/xmlactions">Read</saml:Action>
          </saml:Actions>
        </saml:AuthorizationDecisionStatment>

        <saml:AuthorizationDecisionStatement
Resource=" "http://medico.com/records/ID1234#xpointer(/record/patient/patientDoB)" Decision="Deny">
          <saml:Subject>
            <saml:NameIdentifier SecurityDomain="foo" Name="baa"/>
          </saml:Subject>
          <saml:Actions>
            <saml:Action Namespace="http://www.oasis-open.org/…/xmlactions">Read</saml:Action>
          </saml:Actions>
        </saml:AuthorizationDecisionStatment>
      </saml:Assertion>
    </Response>
```

In the above example, we assumed that the first authorization decision statement is the decision to the document which identifier is ID1212, and that the second authorization decision statement is the decision to the document which identifier is ID1234. This mapping definition depends on each application.

2.2 Simple Path Resource Reference

Potential problem of using XPath expression to point target node(s) would be that the PDP must retrieve the target XML instance document from the XML repository because most of the commercial XPath processors require XML instance document in evaluating the XPath expression. If the target XML is huge, it is not wise to retrieve each XML instance. Then, we use a simple path expression instead of rich XPath expression. The simple path expression is based on popular path expressions used in e.g. file systems that just uses “/” as a path separator.

```
<AuthorizationDecisionQuery Resource=”http://medico.com/records?/record/patient/patientDoB)”>
…
</AuthorizationDecisionQuery>
```

Above query has a simple path expression “/record/patient/patientDoB”. This path expression may satisfy the path expression written in the policy like “/record/*/patientDoB”. We do not specify here the syntax and the semantics of the simple path expression in detail.

3. XACML Policies and Resource Extensions

This section shows two XACML extension examples that make policy specifications much more concise and facilitate policy authoring.

3.1 Example 1: Resource Extension using XPath

Figure 3 shows how to write resource-dependent functions and predicates. XPath expression is used in the resource attribute in SAML authorization decision request as Figure 3 shows.

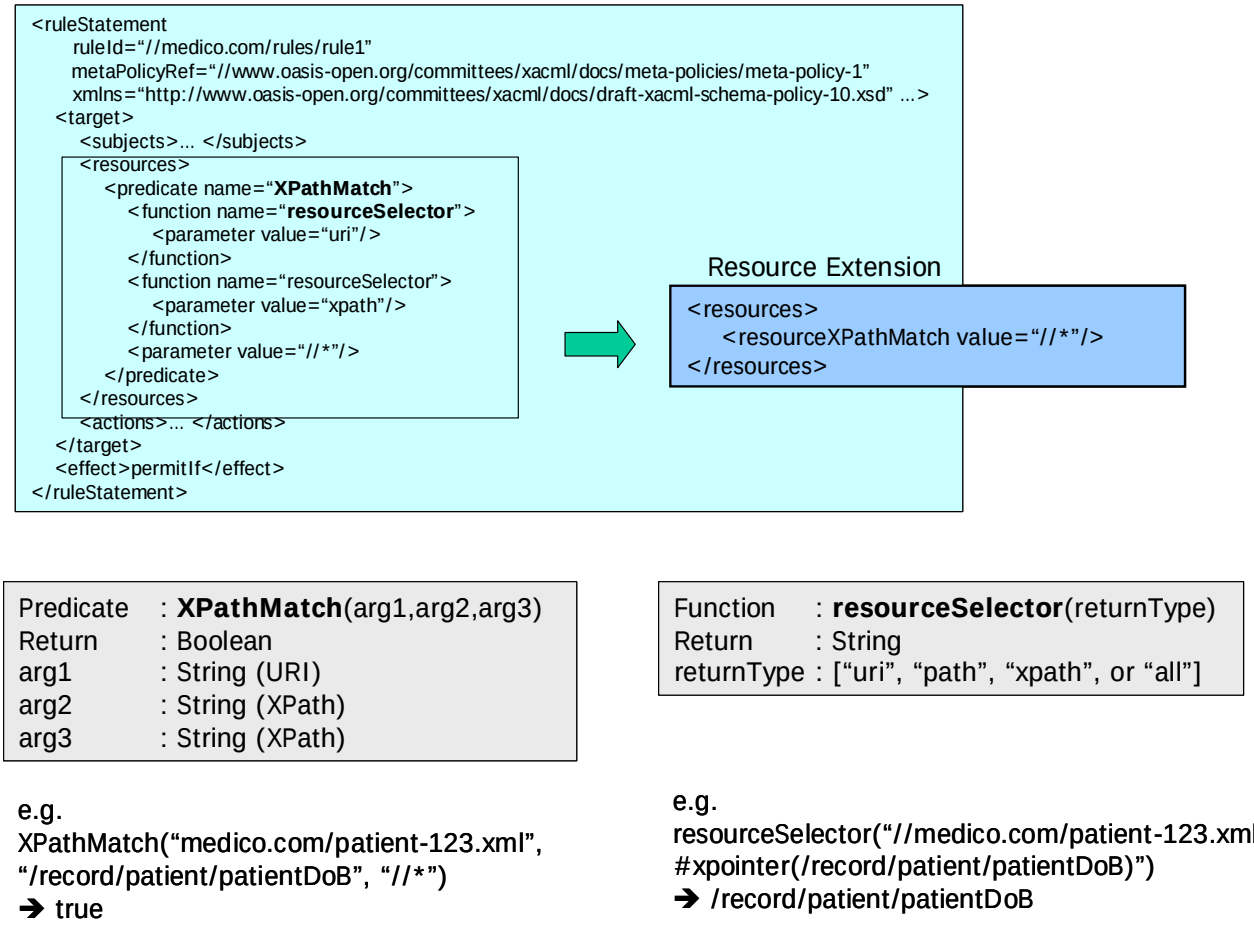

Figure 3. Resource extension using XPath expression

In the box of resource extension, <resourceXPathMatch> element indicates a potential resource extension (or macro) that provides the same meaning as the left-hand predicate-based specification. The merit of using resource extension is to make policy much more concise and comprehensible to human users.

3.2 Example2: Resource Extension using Simple Path Expression

Figure 4 shows how the simple path expression (refer to Section 2.5) is used in the resource attribute in SAML authorization decision request.

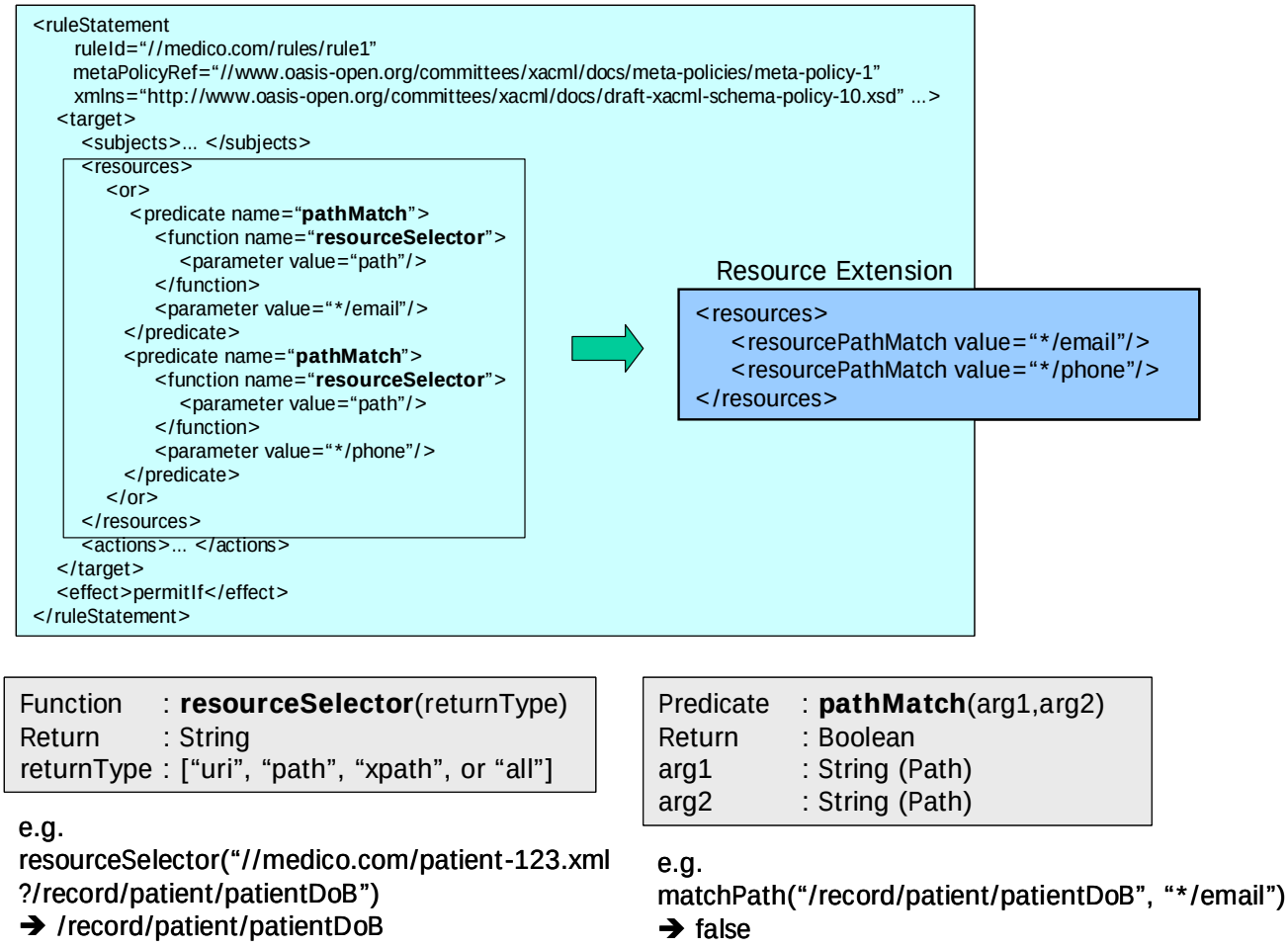

Figure 4. Resource extension using simple path expression.

3.3 Potential Extension Points

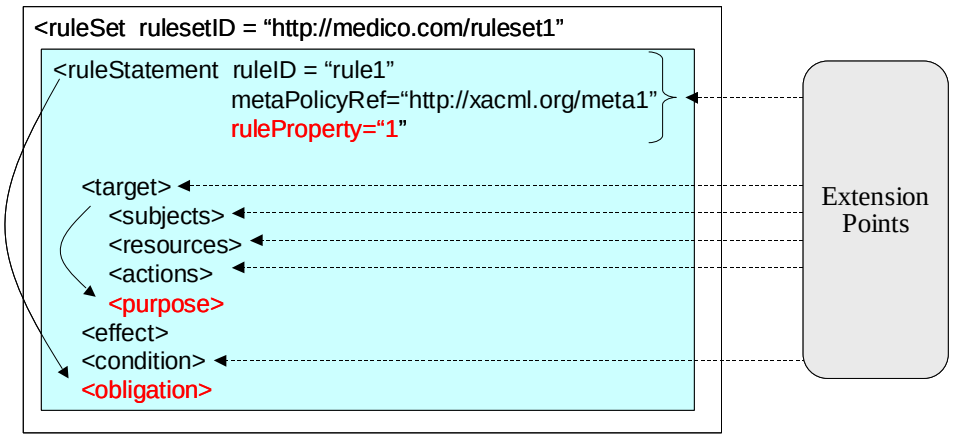

Figure 5. Potential Extension Points

4. XACML Policy Evaluation

In this section, we describe two access control policies that use different semantics of policy evaluation. The first policy uses XACML pre-defined meta policy "meta-policy-1". The second policy uses a user defined hierarchy-based meta policy "meta-policy-hierarchy". It shows how user defined meta-policy is incorporated in XACML policy rules.

4.1 Policy Evaluation Semantics using Meta-Policy-1

Figure 6 shows overview of XACML policy evaluation flow in PDP.

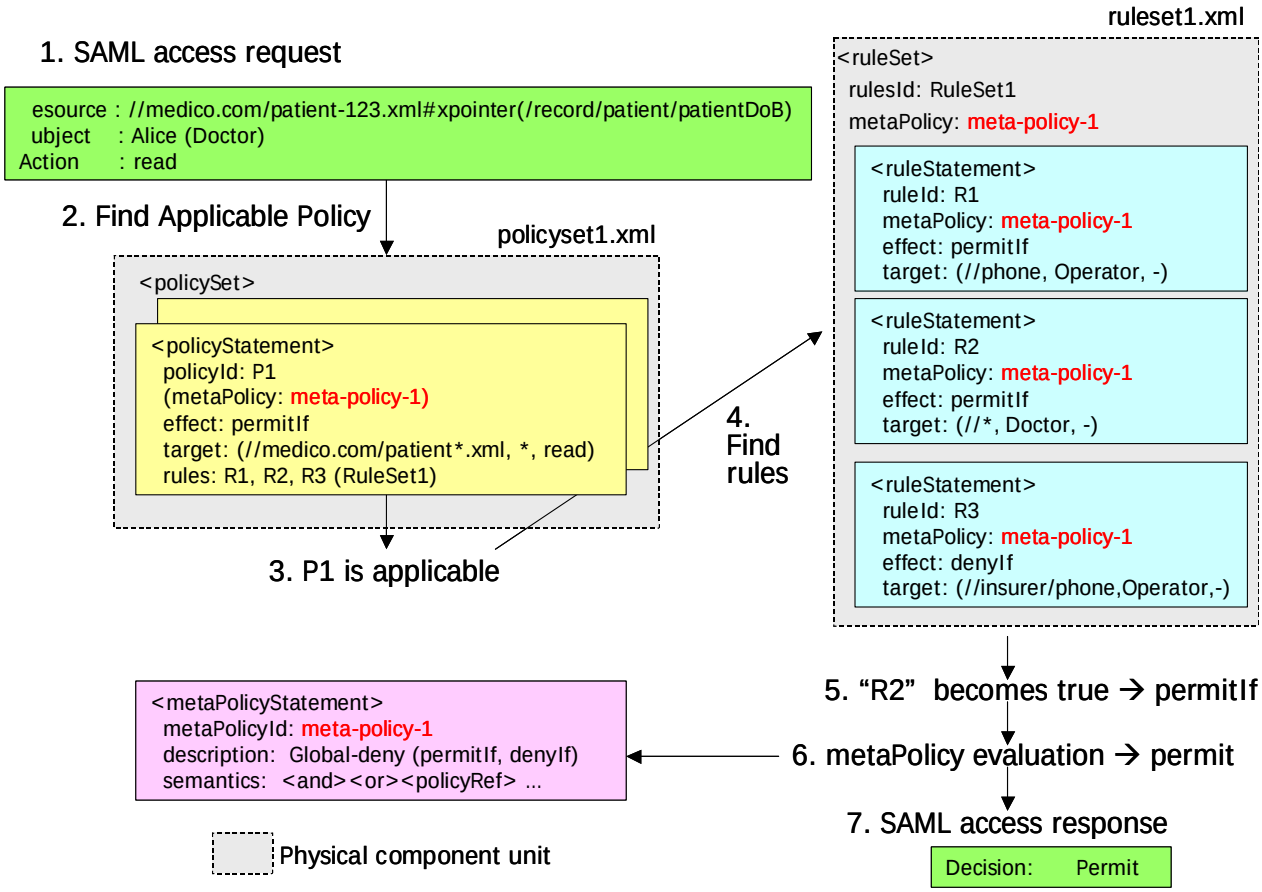

Figure 6. Access evaluation flow using Meta-Policy-1

The following descriptions explain the evaluation process of the PDP.

Step 1. The PDP receives SAML authorization decision request. The request is to access a patientDoB element in the http://medico.com/patient-123.xml file. This request is equivalent to the request described in the example 3.1.

Step 2. PRP/PAP retrieves a set of policies that is applicable to the access request. Target in policyStatement is used to select an applicable policy using just checking parameters in SAML access request.

Step 3. The PRP/PAP finds “P1” is an applicable policy.

Step 4. The PRP/PAP finds that the P1’s subsidiary rules are “R1”, “R2” and “R3”, which are stored in ruleset1.xml. The PRP/PAP send them to the PDP.

Step 5. The PDP evaluates a set of rules and finds that “R2” becomes true. The PDP computes the result based on the meta-policy “meta-policy-1”. In the case where Staff submits an access request on /patient/issuer/phone element, the target parts of “R1” and “R3” are satisfied. The effect of the R1 is “PermitIf” and the effect of the R3 is “DenyIf”. Since the meta-policy-1 means denials always take precedence over permits, the final decision is determined as “Deny”.

Step 6. Since the R2’s effect is “permitIf” and there is no rule that holds, the meta-policy-1 generates “Permit” as a final decision.

Step 7. The PDP generates a SAML authorization decision assertion.

4.2 Policy Evaluation Semantics using Meta-Policy-Hierarchy

Figure 7 shows overview of XACML policy evaluation flow in PDP.

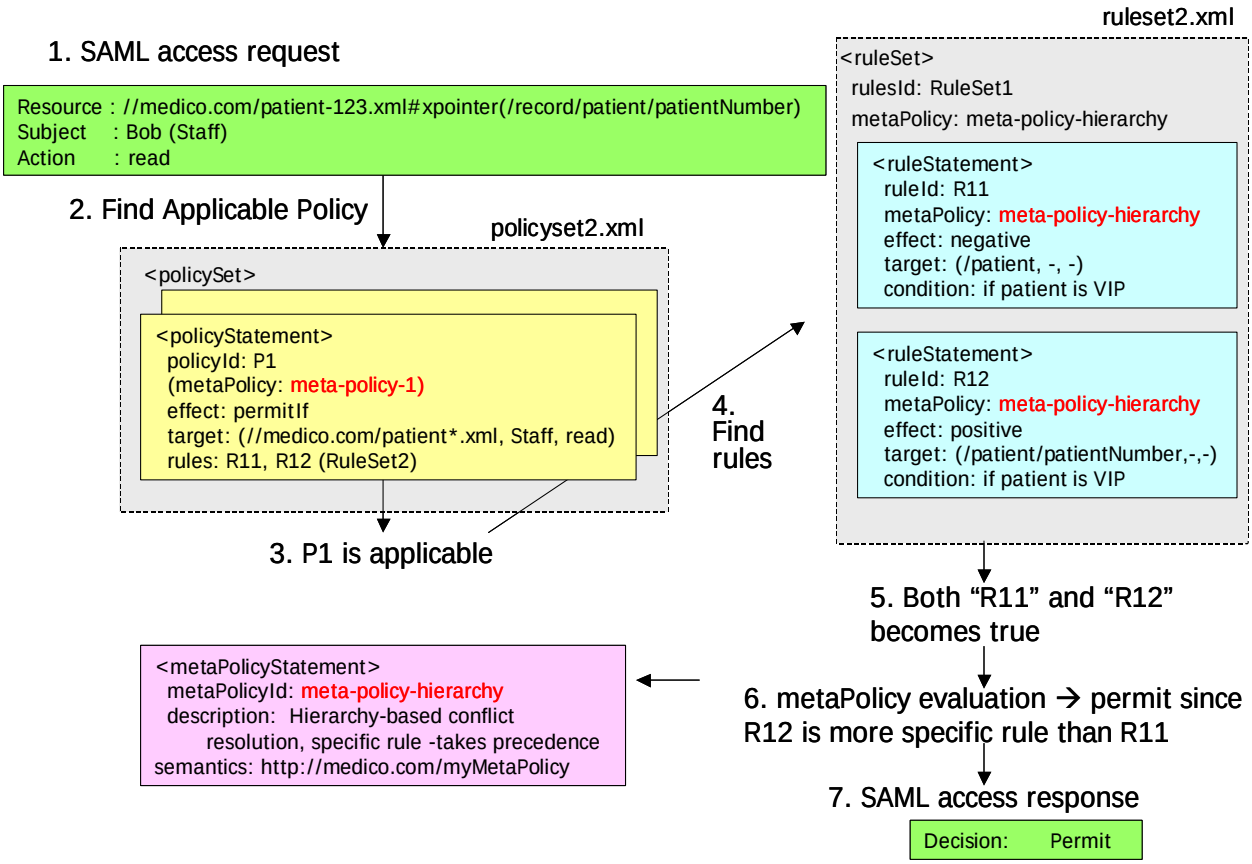

Figure 7. Access evaluation flow using Meta-Policy-Hierarchy

The following descriptions explain the evaluation process of the PDP.

Step 1. The PDP receives SAML authorization decision request. The request is to access a patientNumber element in the http://medico.com/patient-123.xml file.

Step 2. PRP/PAP retrieves a set of policies that is applicable to the access request. Target in policyStatement is used to select an applicable policy using just checking parameters in SAML access request.

Step 3. The PRP/PAP finds that "P1" is an applicable policy.

Step 4. The PRP/PAP finds that the P1's subsidiary rules are "R11" and "R12", which are stored in ruleset2.xml. The PRP/PAP send them to the PDP.

Step 5. The PDP evaluates a set of rules and finds that both the R11 and the R12 become true since the patient is VIP. Both rules are conflicting because the R11 denies while the R12 grants,

Step 6. Then the PDP calls the user-defined meta-policy “meta-policy-hierarchy” by URI. The meta-policy determines grant decision because the rule R12 is more specific rule than the R11.

Step 7. The PDP generates a grant SAML authorization decision assertion.

Reference

[1] XACML, The OASIS extensible Access Control Markup Language (XACML), Committee Draft Version 1.0, 8 March, 2002, http://www.oasis-open.org/committees/xacml

[2] SAML, The OASIS Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML), Committee Working Draft, Version 27, February 14th, 2002, , http://www.oasis-open.org/committees/security

[3] XPointer, XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation, 11 September 2001, http://www.w3.org/TR/2001/CR-xptr-20010911.

[4] URI, RFC 2396.

[5] XPath, XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/xpath.