Using XACML for Privacy Control in SAML-Based Identity Federations

Wolfgang Hommel

Munich Network Management Team,
Leibniz Computing Center Munich
hommel@lrz.de

Abstract. With Federated Identity Management (FIM) protocols, service providers can request user attributes, such as the billing address, from the user's identity provider. Access to this information is managed using so-called Attribute Release Policies (ARPs). In this paper, we first analyze various shortcomings of existing ARP implementations; then, we demonstrate that the eXtensible Access Control Markup Language (XACML) is very suitable for the task. We present an architecture for the integration of XACML ARPs into SAML-based identity providers and specify the policy evaluation workflows. We also introduce our implementation and its integration into the Shibboleth architecture.

1 Introduction

With Identity & Access Management (I&AM) systems, organizations are able to efficiently manage their employees' and customers' Personally Identifiable Information (PII) and access rights to local services, typically by storing them in a central enterprise directory or relational database.

To support cross-organizational business processes, I&AM has developed into Federated Identity Management (FIM); FIM standards, such as the Security Assertion Markup Language (SAML, [1]), enable cross-domain Web single sign-on, i.e. users are being authenticated by their so-called identity provider (IDP) and may then use external service providers (SPs) without requiring separate accounts there. Instead, the SPs trust the IDP, and the IDP vouches that the user has successfully been authenticated. A set of SPs and IDPs with such trust relationships established is called an identity federation.

The FIM protocols do not only provide single sign-on capabilities, i.e. the transmission of authentication information, but also support the exchange of user attributes between SP and IDP. For example, a SP could request a user's billing address and credit card information from the IDP. In business-to-business (B2B) scenarios, the IDP typically is the organization the user is working for, while in business-to-customer (B2C) scenarios it could be the user's ISP or credit card company.

Obviously, access to sensitive data such as Personally Identifiable Information (PII) must be restricted, i.e. there must be a way to control which attributes the IDP hands out to the SPs, in order to protect the user's privacy and thus gain the user's acceptance. While the necessity for such a control mechanism is well-known under the term *Attribute Release Policies* (ARPs), none of the three major FIM standards — SAML [1], Liberty Alliance [2] and WS-Federation [3] — addresses this issue concretely and instead leaves it up to actual implementations.

Fortunately, although not required for standards compliance, some FIM implementations offer ARP support; e.g., Shibboleth [4] is the most advanced and wide-spread open source FIM software currently in use, with its focus on privacy clearly being one of the major reasons for its popularity. However, even Shibboleth only provides rudimentary ARPs in a proprietary format, and the development of more sophisticated ARPs is explicitely encouraged. We analyze existing ARP implementations and their deficiencies in section 2.

The *eXtensible Access Control Markup Language* (XACML, [5]) is a generic and very flexible language for modeling access rights. In section 3, we derive XACML's suitability for the formulation and enforcement of ARPs and demonstrate how it fulfills an advanced set of ARP design criteria and goals. We present an architecture for the integration of XACML ARPs into SAML-based identity providers and then introduce our implementation for Shibboleth in section 4.

2 Related Work and State of the Art

Privacy on the internet and in e-commerce scenarios is a well-studied field and several solutions have found many adopters. To clarify the scope of our work, we first demonstrate how our intents differ from and complement those found in the established privacy standard P3P [6]. We then analyze two ARP implementations and show their limits by means of an e-commerce scenario.

Independent from the development of the FIM standards, the *Platform for Privacy Preferences* (P3P) has been standardized by the W3C for the use in web sites. P3P-enabled web browsers automatically fetch a web site's privacy policies; by comparing them with the user's locally specified preferences, they can decide whether the user agrees to use the site under the given privacy conditions. P3P is neither intended nor suitable for modeling FIM ARPs, because it is an SP-side-only mechanism which does not specify how user preferences shall be stored on the browser or IDP side. It also is limited to web sites and defines an e-commerce specific user profile, whereas FIM protocols work for any kind of web service and support federation-specific user attributes. However, our XACML approach to FIM ARPs leverages the rationale behind P3P and the P3P Preference Exchange Language (APPEL, [7]).

Shibboleth [4] is based on SAML and due to its origin, the higher education institutions in the USA, privacy is an important aspect, so its built-in support for fine-grained ARPs comes at no surprise. Shibboleth distinguishes between *site ARPs*, which are used by IDP administrators to specify defaults for all users, and individual *user ARPs*. Shibboleth ARPs consist of *rules*. Each rule specifies one *target*, i.e. a tuple *(service provider, service)*, which allows to differentiate between multiple services offered by the same SP. For each *attribute* in the rule, this target's access can be granted or denied, optionally based on the attribute's current value. In the following example access to the user's surname is granted to every SP (XML namespaces have been removed to enhance readability):

```
<AttributeReleasePolicy>
    <Rule>
        <Target> <AnyTarget/> </Target>
        <Attribute name="surname">
            <AnyValue release="permit"/>
        </Attribute>
    </Rule>
</AttributeReleasePolicy>
```

Shibboleth combines the site ARP and the user ARP to form the *effective ARP*; if there is a conflict, i.e. one ARP allows access to an attribute while the other does not, or if the SP requests an attribute for which no ARP has been defined, access to the attribute will be denied.

To support distributed management of site ARPs and to distinguish between multiple roles a user can be acting in, Nazareth and Smith suggested an alternative implementation, which uses public key based ARPs [8]. They are choosing the simple public key infrastructure (SPKI, [9]) and the simple distributed security infrastructure (SDSI, [10]) as a base for their ARPs. This approach features hierarchical ARPs, so, for example, a department's ARP can be intersected with the whole company's ARP to form the resulting site ARP, which in turn is intersected with the user ARP. Opposed to Shibboleth's built-in ARPs, no conditions on an attribute's current value can be specified.

Both implementations lack functionality which is demanded in many real world scenarios; those deficiencies are:

– The ARPs are not context sensitive. For example, users typically are willing to grant access to more attributes, such as their credit card data, when they purchase something from a web site than when they are just looking for information; i.e., the purpose why the SP requests the attribute is not considered at all.
– No obligations can be specified. As an example, a user might want to be informed whenever an SP accesses the credit card data, e.g. by means of a logfile or by email.
– Access must be granted or denied to each attribute separately, i.e. there is no way to group attributes. For example, a delivery address may consist of the attributes *given name*, *surname*, *street*, *postal code* and *city*. It is cumbersome having to set up five rules per target instead of one.
– The access conditions are not flexible enough. For example, only the currently requested attribute's value can be part of a Shibboleth ARP condition and there are no environmental functions available; so, if credit card number and expiry date are stored in separate attributes, there is no way to release the credit card number only if it is still valid.

Both approaches also use proprietary formats, leading to the typical implications such as lacking interoperability and the need for dedicated tools as well as additional implementation work.

3 XACML-Based Attribute Release Policies

We will now demonstrate that XACML is an excellent choice to model and enforce ARPs; our architecture, which integrates XACML components into a SAML-based IDP, is introduced in section 3.2. We tailor XACML to specify the ARP syntax and semantics in section 3.3 and define the policy evaluation workflow in section 3.4.

3.1 XACML's Suitability

There are many organizational and technical reasons to use XACML:

1. *Interoperability.* XACML is an OASIS-ratified standard which has successfully been employed in distributed access control before, for example in combination with SAML ([11], [12]) and PERMIS [13]. Its relationship to P3P has been outlined in [14]. Developers and administrators do not need to learn yet another policy language, and GUIs for end users might be re-used with only minor modifications.
2. *Compatibility.* As both are XML-based, Shibboleth ARPs can easily be converted to XACML ARPs. The algorithm is outlined in section 4.
3. *Extensibility.* As requirements are known to change over time and as users will be more familiar with ARP concepts, the language used for ARPs must be flexible enough to allow later extensions; XACML clearly is.
4. *Schema independency.* Opposed to e.g. P3P, XACML has not been designed for a fixed schema; instead, each identity federation can select a suitable data schema or create a dedicated new one. Due to XACML's support for XPath expressions, attributes need not be flat key/value pairs, but structured attributes are also supported. Note that a standardized format for ARPs is independent of the arbitrary format of the data protected by ARPs.
5. *Multiple roles.* IDPs may allow a user to store several profiles, e.g. one used at work and one used in spare time; XACML ARPs can easily be applied to each of them.
6. *Grouping of attributes.* XACML allows the definition of variables, which can be used to group attributes, so access rules need not be specified for each attribute separately. An example is given below.
7. *Decentralized management.* Besides distinguishing between *user ARPs* and *site ARPs*, it is possible to split ARPs into multiple distributed parts, each of which can be maintained on its own. The distribution optionally can reflect hierarchical structures, but priority based and other policy conflict resolution mechanisms are supported as well. The total number of rules required even for sophisticated policies can be kept low. Policy evaluation is easy to understand for the users, and the results are comprehensible.

8. *Conditions.* XACML is very flexible regarding the formulation of conditions under which an attribute can be accessed. Primarily, this includes the specification of a) the *service provider* who requests the data, b) the actual *service* being used, in case an SP offers more than one service, and c) the *purpose* the data is being collected for. Furthermore, all attributes' current values and environmental information, e.g. the current date and time, can be used.
9. *Obligations.* XACML features the specification of obligations, such as sending an email or writing to a logfile when a positive or negative decision about an access attempt has been made.
10. *Optional use of PKI.* While it is possible to use an existing public key infrastructure (PKI) to assure the integrity of user ARPs, it is not a prerequisite for the use of ARPs. In particular, users are not required to handle client-side certificates with their web browser, as this is often error-prone and constrains the use of different machines, devices and browsers. Note that this only affects how ARPs are stored and is independent of whether the released attributes are transmitted to the service provider encrypted or not.
11. *Existing implementation.* XACML ARPs can be evaluated by any standard compliant XACML implementation. An excellent open source implementation is available [15].

Yet, XACML is a generic access control language and must be tailored to our purpose. After an architectural overview, we specify the XACML elements, which are necessary for ARPs, along with their syntax and semantics.

3.2 Architectural Overview

We have integrated an XACML component into a SAML-based IDP, which is minimally invasive and maintains full SAML compatibility. Our XACML component consists of a policy enforcement point (PEP) which we have designed and implemented as described below, and an out-of-the-box XACML policy decision point (PDP).

Figure 1 shows a high-level overview of the relevant components:

– Attribute requests are received by the SAML PDP, which passes them on to our XACML PEP.
– The XACML PEP converts attribute requests into appropriate XACML requests, which the XACML PDP evaluates. Details are given below.
– The attribute values and ARPs are kept in dedicated stores, such as LDAP servers or relational database management systems.
– Administrators and users use dedicated interfaces to maintain the site and user ARPs, respectively. For users, the ARP editing frontend could be combined with the usual self services, i.e. the web site where they can change their passwords, set up their e-mail addresses, update their personal information, etc. The realization of a suitable web interface will be part of our future work.

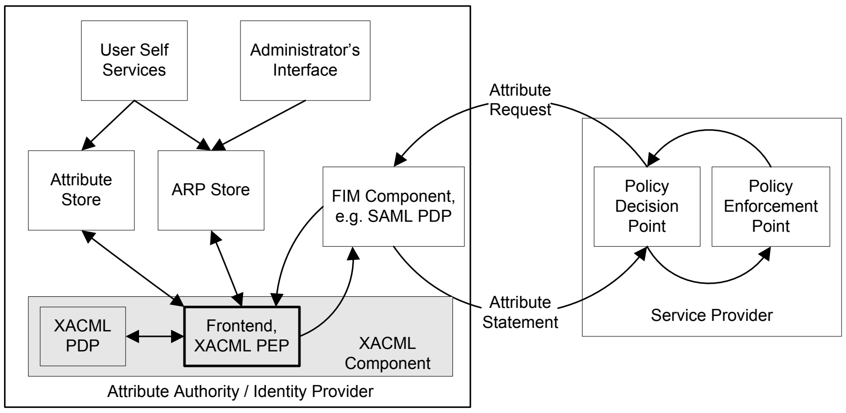

Fig. 1. Overview of components involved in ARP processing

The ARP processing workflow for new attribute requests is as follows:

1. Several parts of the attribute request are extracted by the SAML PDP and forwarded to the XACML PEP: a) The list of the requested attributes, and b) meta-data, such as an identifier of the service provider, the actual service being used and the purpose as stated by the requester.
2. The XACML PEP creates one XACML request per requested attribute, which is then evaluated by the XACML PDP. This is necessary for the following reason: if the complete list of requested attributes would be passed on to the XACML PDP in a single XACML request, the result would be an “all or nothing” response. This means that if just one attribute was not allowed to be released, none of the requested attributes would be released. However, in practice many SPs are greedy and request more attributes than would be required for service provision. Thus, we have to decide about the access to each of the attributes separately.
3. To provide everything the XACML PDP needs, the PEP fetches the necessary ARPs and attributes from the appropriate stores:
   – Multiple ARPs may have to be evaluated; typically, there is at least one site and one user ARP involved. Their combination and evaluation is specified in section 3.4.
   – Besides the attributes which have been requested, additional attributes for the evaluation of conditions within ARPs may be required. Those attribute values are included in the XACML request as `ResourceContent`, see section 3.3.
4. Each attribute request is then evaluated by the XACML PDP; its response is composed of the release decision and optional XACML obligations. The XACML PEP fulfills these obligations before returning the attributes, whose release was permitted, to the SAML PDP.
5. The SAML PDP delivers the attributes to the SP.

The next section describes the elements available within each XACML ARP.

3.3 XACML ARP Syntax and Semantics

In general, each XACML `policy` consists of `rules`. Rule combining algorithms such as "first applicable" or "deny overrides" control how rules are evaluated and when rule processing stops. Both rules and whole policies can specify `targets`; if the policy's targets do not match the actual attribute requester, none of its rules are considered. An empty `target` definition makes sure that the whole policy is always considered.

Each `rule` must have an effect, which is either `permit` or `deny`. It can declare its own target by specifying the protected resources, one or more subjects and the actions attempted by these subjects, and optionally also have a condition.

An XACML ARP will typically contain the following elements (a complete example can be found below):

1. *Priority specification.* The policy's priority is specified as XACML `CombinerParameter` element. Typically, user ARPs will have higher priorities than site ARPs, so users can override the default settings made by the IDP administrator. The combination of multiple ARPs during the evaluation of a request is described in section 3.4. Lines 2–6 demonstrate the priority declaration in the example.
2. *Rule precedence specification.* Each policy must choose one rule combining algorithm. XACML's built-in "first applicable" algorithm, which stops rule evaluation after the first matching rule has been found, is suitable for most tasks and easy to comprehend by the users (see line 1 in the example).
3. *Grouping of attributes.* To group attributes, the names of any number of attributes can be concatenated to form a regular expression, e.g. `Street|-ZIP|City`, and assigned to a variable using a `VariableDefinition` element.
4. *Attribute specification.* XACML `resource` elements specify the user attribute identifiers. Each attribute identifier is an URI, which shall be composed of the IDP identifier, the user identifier, the user role and the attribute name. XACML `VariableReference` elements can be used to speficy attribute groups. Wildcards can also be used. In the example, lines 11–20 show how a user's *creditCardNumber* attribute is selected.
5. *Requester specification.* The triple (*service provider, service, purpose*) is specified as a a conjunctive sequence of three `SubjectMatch` elements within an XACML `subject` node-set as shown in lines 21–33 of the example.
6. *Action specification.* The obligatory XACML `action` is always `read`, as SAML does not allow write operations by the SP yet (see lines 34–40).
7. *Conditions.* XACML conditions may be used to achieve even finer-grained restrictions. All user attributes are included as `ResourceContent` in the XACML request. A description of the powerful XACML functions which can be used within conditions is out of the scope of this paper.
8. *Obligations.* XACML provides the `Obligation` element; writing to a text file and sending an email are part of the standard, but arbitrary other obligations can be implemented as well (see lines 42–49 in the example).

If a PKI is available, the integrity of ARPs can be protected by applying XML signatures as described in [16]. Below is an example which grants access to the user's credit card number to an online shop only if an actual book order is placed; an obligation specifies that each allowed release must be logged.

```
<Policy id="xacmlARP1" RuleCombiningAlg="first-applicable">
  <CombinerParameters>
    <CombinerParameter ParameterName="ARPpriority">
      100
    </CombinerParameter>
  </CombinerParameters>
  <Description> ARP by user John Doe </Description>
  <Rule id="CreditCardToBookShop" effect="permit">
    <Description> Release credit card number to bookshop </Description>
    <Target>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="string-equal">
            <AttributeValue>
              idp.example.com/johndoe/defaultrole/creditCardNumber
            </AttributeValue>
            <ResourceAttributeDesignator AttributeId="resource-id" />
          </ResourceMatch>
        </Resource>
      </Resources>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="string-equal" AttributeValue="shop.example.com">
            <SubjectAttributeDesignator AttributeId="service_provider" />
          </SubjectMatch>
          <SubjectMatch MatchId="string-equal" AttributeValue="bookshop">
            <SubjectAttributeDesignator AttributeId="service" />
          </SubjectMatch>
          <SubjectMatch MatchId="string-equal" AttributeValue="purchase">
            <SubjectAttributeDesignator AttributeId="purpose" />
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Actions>
        <Action>
          <ActionMatch MatchId="string-equal" AttributeValue="read">
            <ActionAttributeDesignator AttributeId="action-id" />
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Obligations>
      <Obligation Id="Log" FulfillOn="Permit">
        <AttributeAssignment Id="text">
          Your credit card number has been released to:
          <SubjectAttributeDesignator AttributeId="service_provider" />
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </Rule>
  <Rule id="DoNotReleaseAnythingElse" effect="deny"/>
</Policy>
```

3.4 Policy Evaluation Workflow

For the evaluation of an attribute request, an XACML `PolicySet` is created by combining all relevant ARPs, i.e. those ARPs whose `target` element matches the requester. This is handled by our XACML PEP.

Each ARP has a priority, and the XACML `PolicySet` is built by including the ARPs ordered by decreasing priority; the "first-applicable" algorithm is then used for the evaluation of the `PolicySet`. If multiple ARPs have the same priority, the inner order of their inclusion in the policy set is indeterminate; this should be avoided to achieve deterministic evaluation results, unless other techniques are applied to ensure that those ARPs have disjunctive `target` sets. The resulting `PolicySet` can be evaluated by any standard compliant XACML PDP.

Obviously, the complexity of XACML policies and XACML implementations can lead to security vulnerabilities; we address these issues by using Sun's reference XACML PDP implementation and working on easy and intuitive graphical user interfaces, as outlined in the next sections.

4 Implementation and Integration into Shibboleth

We have implemented the XACML component in Java, using Sun's XACML PDP implementation [15], which does not support XACML variables yet, so attribute grouping has to be done by our XACML PEP if necessary.

A standalone version is command line driven and creates the XACML request which is evaluated by the PDP. It also creates the XACML `PolicySet` as described in section 3.4; future versions will take the more elegant approach of implementing a custom XACML policy combiner which supports policy priorities because XACML itself does not yet, but it provides the necessary extension hooks. Besides its usefulness for development, we will use the standalone version to enable users to test and debug their ARPs through a web interface.

An integration into Shibboleth's IDP component (called Origin) is possible by replacing two methods in the build-in attribute resolver: first, `listPossibleReleaseAttributes()` must return the names of the user attributes which should be retrieved, and afterwards `filterAttributes()` has to remove all attributes whose release is not permitted by the ARPs. The user's and service provider's ids are passed to both methods, which provides sufficient information for identifying, combining and evaluating the relevant XACML-based ARPs.

Shibboleth's built-in ARPs can be lossless converted to XACML-based ARPs. Basically, Shibboleth ARP `targets` become XACML `subjects` and Shibboleth ARP `attribute` elements turn into XACML `resources`. As release decisions are made on `attribute` and not on `rule` level in Shibboleth ARPs, each Shibboleth `attribute` is converted into a dedicated XACML `rule`. We have successfully automated this transformation using an XSLT stylesheet.

5 Summary and Outlook

In this paper, we first analyzed existing implementations of Attribute Release Policies (ARPs), which are the core privacy management tool in today's identity federation standards. We have found several shortcomings and described their consequences for real world applications and user acceptance. We then provided arguments to use XACML as base for ARPs, a well-established access control language standard, which has been successfully used in the field of distributed access control before. We presented an architecture for the integration of XACML ARPs into SAML-based identity providers, which remains fully compliant to the SAML standard. The syntax and semantics of XACML ARPs have been specified along with the policy evaluation workflow, which makes use of an out-of-the-box XACML policy decision point. Finally, we introduced our implementation, the way to integrate it into Shibboleth, a popular open source identity federation software, and outlined an algorithm which converts existing Shibboleth ARPs lossless to XACML ARPs.

We are planning to integrate this ARP engine into the next major version of Shibboleth, but for use in a production environment, intuitive graphical user interfaces for the creation, testing and maintenance of these ARPs must be conceived and implemented to hide the complexity from the end users. We will also investigate the use of XACML for the so-called Attribute Acceptance Policies, which are the counterpart to ARPs on the service provider side; similar deficiencies such as yet another proprietary format can be found there presently.

Acknowledgment. The author would like to thank the members of the Munich Network Management (MNM) Team (`http://www.mnm-team.org/`), directed by Prof. Dr. Hegering, for helpful discussions and valuable comments on earlier drafts of this paper.

References

1. Cantor, S., Kemp, J., Philpott, R., Maler, E.: Security Assertion Markup Language v2.0. OASIS Security Services Technical Committee Standard (2005)
2. Varney, C.: Liberty Alliance — Privacy and Security Best Practices 2.0. `http://project-liberty.org/specs/` (2003)
3. Kaler, C., Nadalin, A.: Web Services Federation Language (WS-Federation). `http://www-106.ibm.com/developerworks/webservices/library/ws-fed/` (2003)
4. Erdos, M., Cantor, S.: Shibboleth architecture (v05). http://shibboleth.internet2.edu/docs/ (2002)
5. Moses, T.: OASIS eXtensible Access Control Markup Language 2.0, core specification. OASIS XACML Technical Committee Standard (2005)
6. Reagle, J., Cranor, L.F.: The Platform for Privacy Preferences. In: Communications of the ACM. Volume 42., ACM Press (1999) 48–55
7. Langheinrich, M.: A P3P Preference Exchange Language — APPEL 1.0. `http://www.w3.org/TR/P3P-preferences/` (2002)
8. Nazareth, S., Smith, S.: Using SPKI/SDSI for Distributed Maintenance of Attribute Release Policies in Shibboleth. Technical Report TR2004-485, Department of Computer Science, Dartmouth College, Hanover, HN 03744 USA (2004)
9. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylnen, T.: SPKI Certificate Theory. IETF Proposed Standard, RFC 2693 (1999)
10. Rivest, R., Lampson, B.: SDSI — A Simple Distributed Security Infrastructure. Presented at CRYPTO'96 Rumpsession (1996)
11. Lepro, R.: Cardea: Dynamic Access Control in Distributed Systems. Technical Report TR NAS–03–020, NASA Advanced Supercomputing Division, Ames (2003)
12. Mazzuca, P.: Access Control in a Distributed Decentralized Network: An XML Approach to Network Security. Honors Thesis, Dartmouth College (2004)
13. Chadwick, D., Otenko, A.: The PERMIS X.509 Role Based Privilege Management Infrastructure. In: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies. SACMAT, ACM Press (2002) 135–140
14. Anderson, A.H.: The Relationship Between XACML and P3P Privacy Policies. `http://research.sun.com/projects/xacml/` (2004)
15. Proctor, S.: Sun's XACML implementation. `http://sunxacml.sf.net/` (2004)
16. Anderson, A.: XML Digital Signature profile of XACML 2.0. OASIS TC Committee draft, 16. September 2004 (2004)