Policy Administration Control and Delegation using XACML and Delegent

Ludwig Seitz*, Erik Rissanen†, Thomas Sandholm‡, Babak Sadighi Firozabadi†, and Olle Mulmo‡
*LIRIS, INSA de Lyon, FRANCE
†ISL, SICS Kista, SWEDEN
‡PDC, KTH Stockholm, SWEDEN

***Abstract*— In this paper we present a system permitting controlled policy administration and delegation using the XACML access control system. The need for these capabilities stems from the use of XACML in the SweGrid Accounting System, which is used to enforce resource allocations to Swedish research projects. Our solution uses a second access control system *Delegent*, which has powerful delegation capabilities. We have implemented limited XML access control in *Delegent*, in order to supervise modifications of the XML-encoded XACML policies. This allows us to use the delegation capabilities of *Delegent* together with the expressive access level permissions of XACML.**

I. Introduction

The Swedish research community has a high demand for computational and storage resources. This need is addressed by SweGrid, a nation-wide Grid of computational resources, currently interconnecting 600 nodes at 6 Swedish High-Performance Computing centers with the 10 Gb/s GigaSunet network.

In a process involving peer review, the Swedish National Allocations Committee (SNAC) distributes resource quotas to research projects. These allocations are enforced in real-time by the SweGrid Accounting System (SGAS) [1]. In SGAS, accounting policy can be defined on a per-project basis using XACML [2]. The default access control mechanism in the administrative part of the system is however relatively coarse-grained: as an administrator of a project, you have the full privileges to change any arbitrary portion of the policy. This is not adequate to fulfill the flexibility requirements for example in case of a project consisting of collaborating teams, where you may want to grant the project leaders the administrative right to add or remove members from their respective teams. This type of grant can be expressed as a constrained delegation. An administrative authorization should be able to constrain the content of all authorizations further down in the delegation chain originating from that authorization. This way someone higher up in the authorization hierarchy can put a constraint to which groups the right may spread and limit the contents of the access level authorization being delegated.

The current XACML standard does not support any form of delegation, neither do the available XACML policy administration points support controlled administration of XACML policies. We therefore need a system that can distribute controlled pieces of policy administration rights to local administrators, without exposing the whole policy database to them. This requires controlling modification to XML documents (XACML policies are encoded in XML) and providing delegation capabilities in the access control system used.

The work presented here describes a case study in which we investigate how an authorization management system based on XACML can be extended to use flexible delegation mechanisms.

The rest of this paper is organized as follows: Section II presents related work. In section III we present the environment of our case study and the software tools we used. Section IV gives a brief overview how our implementation works. We present the components of our implementation and their use in section V. We discuss the results of our work in section VI and finally draw a conclusion and present directions for future work in section VII.

II. Related Work

Since XML [3] has emerged as a standard used to structure, store and send data, various approaches have been proposed to control access to XML documents. All approaches presented here consider XML nodes as the most fine-grained objects for which permissions are issued.

The approach called *static analysis* proposed by Murata et. al. in [4] considers only read actions on XML documents. To refer to XML document nodes they use an XPath fragment that restricts predicates to testing equality between XML attributes and constant values.

In [5] Gabillon and Bruno propose XML document access control based on *XSLT*. Their model is also limited to read actions. Their system uses an XPath fragment that limits the node navigation options.

Damiani et. al. propose an XML document access control approach in [6] and [7]. Their approach uses the full XPath language and supports read and write operations. Write operations are subdivided in three categories: *insert*, *delete* and *update*. However, it remains unclear, how their approach detects which node is going to be affected by the write operation, given the original and/or the modified document. Furthermore the prototype of this proposal [1] currently only supports read-operations.

Bertino et al. have published numerous articles on XML document access control [8], [9], [10], [11] and [12]. We discuss the approach presented in [12]. It supports a set of *browsing* (read) and *authoring* (write) actions, which are grouped

[1] Available from http://seclab.dti.unimi.it/ xml-sec

TABLE I

SUMMARY OF THE FUNCTIONALITIES PROVIDED BY DIFFERENT SYSTEMS.

Approach	node matching	supported actions
Murata et al.	subset of XPath	read
Gabillon and Bruno	subset of XPath	read
Daimiani et al.	full XPath	read, write
Bertino et al.	proprietary language	read, write
XACL	full XPath	read, write

hierarchically. The approach uses a proprietary language to refer to elements of an XML document. According to [13] this language is equivalent to a subset of XPath, supporting the use of predicates, with limited comparisons of elements or attributes to constant values. We can see no justification to replace XPath by a proprietary construct, since XPath is standardized and offers a large variety of useful functions for specifying and restricting the nodes to be selected.

Kudo and Hado propose *XACL* as XML access control language in [14]. Their approach seems to support the full functionality of XPath to select XML nodes as access control objects. They have specified *read, write, create* and *delete* actions analogously to Damiani et. al. (where *write* corresponds to *update* and *create* to *insert*).

The Table I summarizes our findings concerning above systems.

None of the present solutions supports delegation, which is a key requirement for being able to spread the load of administration while minimizing the risk of abuse. Furthermore, our problem requires elaborate XPath predicates to restrict the values that a text or attribute node may adopt, and finally our problem clearly requires an approach that supports write actions on XML documents. We therefore decided to design our own approach fulfilling all of these requirements.

III. ENVIRONMENT AND TOOLS

In this section we present the internal structure of SGAS and the tools we used to achieve the goal of controlled policy administration and having delegation capabilities.

A. SGAS

As mentioned in the introduction, the SweGrid Accounting System (SGAS) [1] enforces resource allocations on SweGrid. SGAS comprises a Bank service responsible for managing the allocations, a logging and usage tracking service and a workload manager integration component. The allocation enforcement is governed by a 3-party (user, resource and allocation authority) authorization policy framework, allowing all stakeholders to manage and enforce their local policies. The Bank service is the natural hub for policy administration, enforcement, combination and evaluation in SGAS. Hence, it offers an easily extendable authorization infrastructure, where custom Policy Decision Points (PDP) and Policy Information Points (PIP) may be combined and controlled by a generic Policy Enforcement Point (PEP), and a general purpose Policy Administration Point (PAP). PIPs are used to gather policy attributes that can be used by PDPs to make authorization decision. Each PDP manages its local policy. SGAS provides a standard XACML policy evaluator[2] combined with a persistence layer storing the policy in a native XML Database. The Policy Administration Point is exposed to administrators, typically allocation authorities or research project leaders, through a standard Web services interface, secured by the means of WS-Security [15]. When a new policy is set via this interface, the Service Authorization Management (SAM) interface, the configured PDPs get a chance to either reject the change or to use the new policy, or parts of it, to update their local policies.

B. XACML

The eXtensible Access Control Markup Language (XACML) [2] was developed in order to provide a uniform way of specifying access control policies in XML. Policies comprising Rules, possibly restricted by Conditions, may be specified and targeted at Resources, Subjects and Actions. Resources, Subjects, Actions and Conditions are matched with information in an authorization request context using Attribute values and a rich set of value-matching functions. Rules and Policies may be combined using a few standard Rule- and Policy-combining algorithms. The outcome or Effect of a policy evaluation may be Permit, Deny, NotApplicable (e.g., no Policy with matching target was found, or all Rules evaluate to false) or Indeterminate (e.g. an error occurred while evaluating a Rule). Further, a Permit decision may be attached with obligations. The current XACML standard does not provide support for delegation, neither does the related assertion language standard SAML [16].

C. Delegent

Delegent is an authorization server based on research done at the Swedish Institute of Computer Science [17]–[20]. The goal of the research has been to enable decentralized management of access control by means of delegation of administration. In Delegent all authorizations are expressed in the form of delegations. A delegation has an issuer, time stamp and an authorization. The interpretation of a delegation is that the issuer asserts the contained authorization.

An authorization can be of two forms, either an access level authorization or an administrative authorization, which are completely separate so neither form implies the other. Access level authorizations allow for the use of the objects under access control and administrative authorizations decide which other authorizations are considered to be valid. We say that one delegation 'supports' another if an administrative authorization in the first delegation allows for the second delegation. The support relations form edges in a graph with the delegations as the nodes. There is a special form of delegation which is used to define the roots of authority in the system. An access level permission is considered to be valid if we can trace a path from it to such a root delegation.

Delegent supports constrained delegation. To do this, Delegent will compare authorizations to see whether one is more

[2]http://sunxacml.sourceforge.net

restricted than the other. This way the mechanism allows for the division of responsibility and the corresponding delegation of authority.

An access level permission in Delegent is a four-way tuple with a subject or group of subjects, an object or group of objects, a method or group of methods and a time interval. This is much simpler than in XACML, so in relation to XACML Delegent offers more features for decentralized administration of the policies, while XACML offers more expressive access permissions.

D. XPath

XPath is a language for addressing parts of an XML document. Its specification [21] was conceived by the W3C and numerous implementations exist [3].

The feature that is important for us in XPath, apart from matching elements of an XML document, is that you can specify conditions, called *predicates* in the XPath specification, on the values of attributes and text nodes in any part of an XML document. Given the fixed structure of XACML policies, we can create XPath expressions that match elements in a specific rule of a specific policy, having specific attribute and text-node values.

E. X-diff

When comparing a policy against its update we need to find the differences between them, independent of reordering, at node level. The general problem of change detection in unordered trees is known to be NP-complete. Luckily certain features of XML can be used to narrow this problem down to a manageable task. We use X-diff [22], a polynomial time algorithm that detects changes in XML documents.

The X-diff algorithm takes an original and a modified XML document as input and generates a third combined XML document that contains processing instruction nodes (a special kind of XML node for passing data to applications), pointing out *updated*, *added* or *deleted* parts of the document. Both deleted and added parts are contained in the combined document. For updated parts the processing instruction contains the original value of the node, while the combined document shows the updated value.

The example in figure 1 shows an X-diff result on a policy update where the effect of the *SomeRule* was altered, a new *Subject* was inserted and the empty Rule *FinalRule* was deleted. [4]

```
<Policy PolicyId="TestPolicy">
   <Rule Effect="Deny" RuleId="SomeRule">
   <?UPDATE Effect FROM "Permit"?>
      <Subject>
      <?INSERT Subject?>
      [contents of the Subject node] ...
      </Subject>
      ...
   </Rule>
   <Rule Effect="Deny" RuleId="FinalRule">
   <?DELETE Rule?>
   </Rule>
</Policy>
```

Fig. 1. An X-diff result on a policy update.

IV. System Overview

This section gives an overview of the system components and elaborates on how they interact. The internal functionality of the components is discussed in detail in section V. We use figure 2 to visualize our explanations.

A typical system task starts at the XACML-PAP. Note that the PAP is not a part of our system, the Policy Administration Control (PAC) is completely agnostic to the XACML-PAP used to update policies. This is made possible by the SAM, to which the XACML-PAP submits requests for policies and for policy updates. When confronted with a policy update request (step 1 in the figure), the SAM contacts the PAC and submits the policy that would result from the requested modifications (step 2). The PAC then gets the original XACML policy from the policy database (step 3) and submits both policies to X-diff, getting back a difference-document (step 4). This difference-document is then parsed by the PAC and each elementary modification is turned into a Delegent access query (step 5). The PAC collects the results of the Delegent-PDP. If any of them is negative it notifies the SAM that the whole update is denied, together with an error message that describes why the update request was rejected (step 7). If no elementary modification is rejected by the Delegent PDP, the whole update is permitted and the PAC writes the modified policy to the XACML policy-database (step 6) and returns a positive result (step 7) to the SAM. The SAM forwards the PAC's results to the XACML-PAP (step 8). The Delegent PAP is independent of all other components and access to it gives unlimited power of Delegent permission administration, thus in turn giving indirect unlimited XACML administration rights.

V. Component Usage

This section describes the components of the policy administration control system. Detailing what parts were changed in which way and what the new resulting functionality looks like.

A. Delegent

To enable Delegent to regulate modifications of an XACML policy we had to introduce a new type of Delegent permission.

The unmodified Delegent engine uses *subject, object* and *method* string identifiers to match a request to a permission. The matching method is string equality. In our modified

[3] We use both the libxml2 implementation for the Delegent part and the Xalan-Java implementation for the SGAS part of our work.

[4] Since XACML policies are very verbose we have simplified them to make the examples more readable

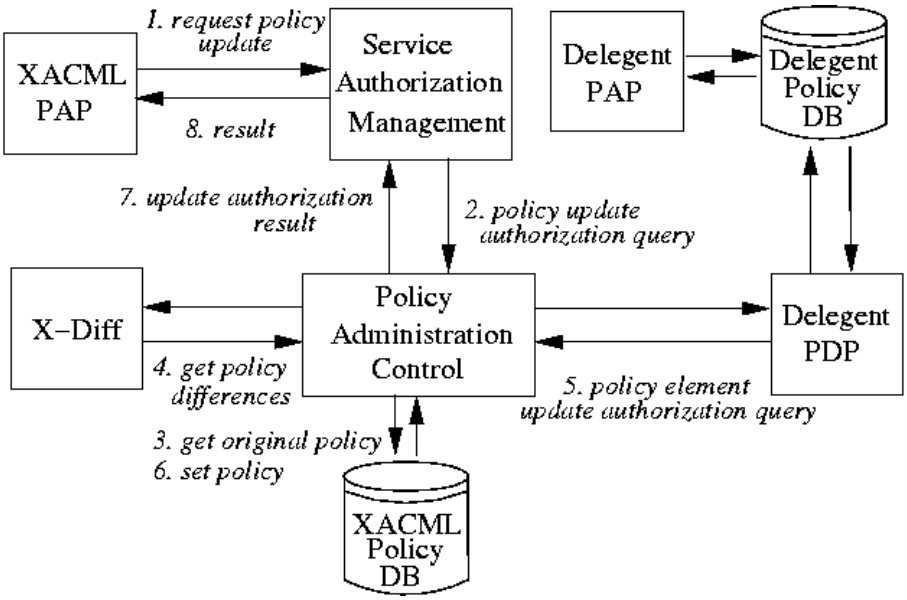

Fig. 2. Overview of the Policy Administration Control system

permission type *subject, object* and *method* are still strings, however, the way the object match is verified has changed as well as the way the object string of an access request is created.

The *access request object* contains the XML structure affected by the modification, and the *permission object* contains an XPath which will be applied to the request object[5]. If one and only one match is found, the permission object applies to the access request object. This implies that the submitter of the access request must ensure that one and only one elementary modification is contained in the request object, otherwise it would be possible to sneak in unchecked modifications together with a permitted one. Our PAC service (see figure2 takes care of splitting the X-Diff results into elementary modifications.

The example in figure 3 shows a Delegent access query and a permission of the type described above. The permission allows *someAdmin* to add any user from the *pdc.kth.se* domain to the target subjects of *TestRule*.

Delegent's constrained delegation mechanism requires that subject, object and method of the delegated permission are restrictions of the same elements in the original permission. As a result of this a delegation will always convey lesser or equal rights. We therefore needed a special method to check whether an object-XPath is a restriction of another.

We have defined the restriction of an XPath in the following way:

Definition 1: An XPath *A'* is a restriction of an XPath *A* if for each node matched by *A'* one of the following applies:

1) The node is also matched by *A*,
2) or the node is a descendant node of a node matched by *A*.

[5]We use the XPath 1.0 standard as specified in [21]

```
<accessQuery>
   <subject>someAdmin</subject>
   <object>
      <Policy PolicyId="TestPolicy">
      <Rule RuleId="TestRule">
      <Subject>
      /O=NorduGrid/OU=pdc.kth.se
      /CN=T.Sandholm
      </Subject>
      </Rule>
      </Policy>
   </object>
   <method>addRuleTargetSubject</method>
</accessQuery>

<permission>
   <subject>someAdmin</subject>
   <object>
      /Policy[@PolicyId="TestPolicy"]
      /Rule[RuleId="TestRule"]
      /Subject
      [starts-with("/O=NorduGrid
                    /OU=pdc.kth.se/")]
   </object>
   <method>addRuleTargetSubject</method>
   ...
</permission>
```

Fig. 3. An example of the new type of Delegent access queries and the corresponding permission.

An exact definition of the *descendant* relation is given in chapter 5 of [21].
This definition can be resumed as the problem of detecting whether an XPath matches a subset of the nodes matched by another. This problem is also known as the *containment problem for XPaths*. In [23] it is shown that the containment problem is co-NP-complete. We have therefore decided to go for a simplified algorithm that may falsely reject that an XPath is a restriction of another, if certain conventions about creating restricted XPaths are not respected.

These conventions are the following:

Definition 2: A restriction *A'* of an XPath *A* must copy the entire content of *A* with the following three exceptions:

- *A'* may add predicates anywhere in *A*.
- *A'* may add commands at the end that match successor nodes of those matched by *A*.
- If *A* contains one or more *or*-operators, *A'* may leave any of them out together with one of their operands.

We can then use a simple character matching algorithm that takes care of the three special cases above to check if an XPath is a restriction of another.

B. XACML

XACML was not modified at all by our system. However, certain parts of the XACML specification are not supported yet (policy sets and obligations). By putting up the SAM as an interface between the XACML-PAP and the PAC, XACML can remain unaware of the functions of the PAC and can be used the same way with or without it.

C. X-diff

X-diff itself has been slightly modified to add the required interfaces (in its original version it provided only a file-in-file-out interface). However more importantly a post-treatment of the X-diff results has proven to be necessary.

Given a policy update that includes two modifications, one deleting an element in a policy and the other adding a new element of the same type as the deleted one in the same policy, X-diff returns the result that the deleted element was updated. This is semantically incorrect from the access control point of view (e.g. deleteing a user and adding a new one is not the same as updating one user and may require different permissions). Such an update should be treated as separate delete-element and add-new-element modifications. The example in figure 4 shows such a semantically incorrect X-diff result and the final result after passing our semantic post-treatment.

D. PAC

The PAC collects the semantically corrected results of X-diff. For each of those results a Delegent access query is generated. PAC uses the X-diff results to find the access query object. The access query subject is passed down to the PAC from the SAM. The access query method is determined by the object (e.g., if the object is a modification of a rule-subject the method becomes *modify-rule-subject*). Those Delegent queries are submitted to the modified Delegent-PDP and the PAC collects the results. If none of them are negative, the updated policy is written back to the XACML policy database; otherwise an error message describing the reason why the update request was rejected is sent back to the XACML-PAP through the SAM.

```
Semantically false version:

<Subject>
   /O=NorduGrid/OU=pdc.kth.se
   /CN=O.Mulmo
<?UPDATE FROM
   /O=NorduGrid/OU=pdc.kth.se
?>
</Subject>

Corrected version:

<Subject>
<?INSERT Subject?>
   /O=NorduGrid/OU=pdc.kth.se
   /CN=O.Mulmo
</Subject>
<Subject>
<?DELETE Subject?>
   /O=NorduGrid/OU=pdc.kth.se
   /CN=T.Sandholm
</Subject>
```

Fig. 4. A semantically false X-diff result and the corrected version.

VI. Results

In this section we summarize the results of our work and describe what our solution can and cannot do.

- The system makes it possible to use the powerful XACML access level permissions together with the delegation mechanisms provided by Delegent. Delegation permissions are described in the Delegent policy, whereas the XACML policy describes the access level permissions.
- The system makes it possible to control access to the XACML policy database by re-routing the XACML-PAP output through the PAC module and checking it against Delegent policy access control rules.
- The system relies on the SAM for authentication of a user issuing a request through an XACML-PAP.
- The system does not control access to the Delegent policy database.
- The system makes the assumption that communication between the different subsystems is secured (e.g., by TLS/SSL)

VII. Conclusions and Future Work

With the present solution we can prevent abusive modification of the XACML policy by administrators having access to the XACML PAP. However, access to the Delegent PAP needs to be restricted since it is not protected and the Delegent policy controls the allowed modifications of the XACML policy. The system can therefore be used by the primary SGAS administrators to create a Delegent policy, specifying allowed modifications on different elements of an XACML policy for different users. Then access to the XACML PAP can be granted to the administrators of local sites, which have been allocated accounts in the SGAS system.

Integrating the PAC system into SGAS is relatively simple, since it only interacts with the SAM. However, it requires a deployment of a modified version of Delegent and no inherent protection of the Delegent PAP exists. Furthermore, managing the hybrid system involving two different access control mechanisms and two different policy databases is somewhat cumbersome and can therefore only be a temporary solution. A good solution to this problem would be to add delegation mechanisms to XACML, which then could be used to control access to its own policies. Recently the XACML community has started to work on such approaches [24]. However, this is outside the scope of this case study.

The PAC system could easily be extended to also manage and restrict read access of the XACML policies, and with a little more effort to a general read/write access control system for XML documents supporting permission delegation. We believe delegation is a crucial functionality when controlling access and it should not be neglected.

Aknowledgement

The authors would like to thank the scientific service of the French embassy in Stockholm for funding the French-Swedish cooperation that led to this work.

References

[1] Sandholm, T., Gardfjäll, P., Elmroth, E., Johnsson, L., Mulmo, O.: An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers. In: Proceedings of the 2nd International Conference on Service Oriented Computing, New York City, USA (2004) 279–288

[2] Godik, S., Eds., T.M.: eXtensible Access Control Markup Language (XACML). Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2003) http://www.oasis-open.org/.

[3] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0. W3C recommendation, World Wide Web Consortium (1998) http://www.w3.org/TR/REC-xml.

[4] Murata, M., Tozawa, A., Kudo, M.: XML Access Control Using Static Analysis. In: Proceedings of the 10th ACM conference on computer and communication security, Washington, DC, USA (2003)

[5] Gabillon, A., Bruno, E.: Regulating Access to XML documents. In: Proceedings of the fifteenth annual working conference on Database and application security, Niagara on the Lake, Ontaria, Canada (2001)

[6] Damani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: Securing XML Documents. In: Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology, Konstanz, Germany (2000) 121–135

[7] Damani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: A Fine-Grained Access Control System. In: Transactions on Information and System Security (TISSEC). Volume 5. ACM (2002) 169–202

[8] Bertino, E., Castano, S., Ferrari, E., Mesiti, M.: Specifying and enforcing access control policies for xml document sources. World Wide Web Journal **3** (2000)

[9] Bertino, E., Castano, S., Ferrari, E.: On specifying security policies for web documents with an xml-based language. In: Proceedings of the sixth ACM Symposium on Access Control Models and Technologies, SACMAT, Chantilly, Virginia, USA (2001)

[10] Bertino, E., Castano, S., Ferrari, E.: Securing xml documents: The author-x project demonstration. In: Proceedings of the ACM Special Interest Group on Management Of Data conference (SIGMOD), Santa Barbara, CA, USA (2001)

[11] Bertino, E., Castano, S., Ferrari, E., Mesiti, M.: Protection and admininstration of xml data sources. Data & Knowledge Engineering (ELSEVIER) **43** (2002)

[12] Bertino, E., Ferrari, E.: Secure and Selective Dissemination of XML Documents. In: Transactions on Information and System Security (TISSEC). Volume 5. ACM (2002) 290–331

[13] Fundulaki, I., Marx, M.: Specifying access control policies for xml documents with xpath. In: Proceedings of 9th ACM Symposium on Access Control Models and Technologies, SACMAT, New York, USA (2004)

[14] Kudo, M., Hada, S.: XML Document Security based on Provisional Authorization. In: Proceedings of the 7th ACM conference on Computer and communications security, Athens, Greece (2000) 87–96

[15] Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R., Eds., P.G.: Web Services Security 1.0. Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2004) http://www.oasis-open.org.

[16] Maler, E., Mishra, P., Eds., R.P.: The OASIS Security Assertion Markup Language (SAML) v1.1. Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2003) http://www.oasis-open.org.

[17] Firozabadi, B.S., Sergot, M., Bandmann, O.: Using Authority Certificates to Create Management Structures. In: proceedings of Security Protocols, 9th International Workshop, Cambridge, UK. (2001) 134–145

[18] Firozabadi, B.S., Sergot, M.: Revocation Schemes for Delegated Authorities. In: proceedings of IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, USA (2002)

[19] Firozabadi, B.S., Sergot, M.: Revocation in the Privilege Calculus. In: Proceedings of the 1st International Workshop on Formal Aspects in Security and Trust (FAST 2003), Pisa, Italy (2003) 39–51

[20] Bandmann, O., Dam, M., Firozabadi, B.S.: Constrained Delegations. In: proceedings of 2002 IEEE Symposium on Security and Privacy, Oakland, CA, USA (2002)

[21] Clark, J., DeRose, S.: XML Path Language (XPath). W3C recommendation, World Wide Web Consortium (1999) http://www.w3.org/TR/xpath.

[22] Yuan, W., D. DeWitt, J.C.: X-diff: An Effective Change Detection Algorithm for XML Documents. In: Proceedings of the 19th International Conference on Data Engineering. (2003) 519–530

[23] Neven, F., Schwentick, T.: XPath Containment in the Presence of Disjunction, DTDs, and Variables. In: Proceedings of 9th International Conference on Database Theory (ICDT). (2003) 315–329

[24] Rissanen, E., Firozabadi, B.S.: Administrative Delegation in XACML. In: Proceedings of the W3C Workshop on Constraints and Capabilities for Web Services, Redwood Shores, CA, USA (2004)