Administrative Delegation in XACML – Position Paper

Erik Rissanen (mirty@sics.se) Babak Sadighi Firozabadi (babak@sics.se)

Swedish Institute of Computer Science
Box 1263
164 29 KISTA
SWEDEN

Abstract

In this position paper we argue the need for mechanisms to support decentralised administration of policies in highly dynamic organisations. We show how current specification of XACML can be extended to support delegation of policies.

Use Case

We base our position paper on the following use case: A web service uses an XACML Policy Decision Point (PDP) for its access control decisions. The service is used by a very large and dynamic organisation, with frequent changes to the access control policy. The rights to change the policy also change frequently.

Introduction

XACML is a highly expressive language for access control policies. The specification of XACML includes the language and its semantics and a framework for making access control decisions based on XACML policies. However, XACML is currently lacking an access control model for the policy itself.

The current XACML model of policy administration puts the access control of the administration of the policies outside the policy model. To control who may edit the policy, mechanisms such as operating system level access control has to be used. In large distributed systems such mechanisms may prove to be difficult to manage. There may be a need to manage the policies in parts of the system not under the control and within the trust of a specific Policy Decision Point, for instance from a mobile device. The rights to change the policy may be highly dynamic themselves. This leads to a need for an access control policy model for the policy itself. Our research has been focused on these issues.

The main result of our research is a framework and a calculus, called privilege calculus, for access permissions and their administrations [4,5]. In this framework, we distinguish between access permissions and administrative permissions, both referred to as privileges. Each privilege in the framework has an issuer and a validity-time. The calculus allows us to deal with both privileges and their administration. The core mechanism of privilege calculus is constrained delegation which allows one to put constraints on how a privilege, access permission or administrative permission, can be created for other users.

It should be pointed out that the delegation mechanism in privilege calculus is administrative delegation, not proxy delegation between tiers in a distributed system. It is about creating new long-term access control policies by means of delegation in a decentralised organization.

It has recently been discussed by a number of XACML TC members to add administrative delegation to XACML. [1,2,3] The ideas discussed within XACML TC are very similar to the delegation mechanism of privilege calculus. We are now looking into the possibility of extending current XACML specification and implementing our delegation model in SUN's open source XACML implementation. This will be part of an ongoing project in which we investigate the use of XACML as a policy language for distributed services in the highly dynamic and decentralised networks needed for Network Based Defence (NBD) scenarios.

In this position paper we outline how we will extend XACML to meed the needs of NBD.

We do not address the use case from the call for papers since it falls outside the scope of policy administration.

Constrained Delegation

Within the privilege calculus we distinguish between administrative permissions and access permissions. An access permission is simply a traditional permission which grants access to a resource. An XACML 1.1 rule is an example of an access permission. Administrative permissions specify what other permissions may be created. XACML does not currently support administrative permissions. In the privilege calculus an administrative permission contains an access permission and a constraint on its delegation. The access permission constrains what accesses may ultimately be allowed based on that administrative permission. The delegation constraint specifies to whom the permission may be delegated. It does so by means of a sequence of constraints that corresponds to a sequence of issuers in a chain of delegation.

The purpose of the constrained delegation is for instance to limit the permissions to a part of an organisation. The constraints can also be used for expressing that someone may administer permissions of others, but cannot grant those permissions to herself, which may be useful in e.g. outsourcing scenarios.

Proposed solution

The solution we will explore includes the following changes to XACML.

Every policy will have an issuer. We assume that policies are digitally signed by the issuer for secure distribution.

We add new structured data-types to express chains of delegation and constraints on delegation. We also add a new function on these types which is able to compare a chain to a constraint.

An access level permission is like an XACML 1.1 rule, with the exception that the policy it belongs to has an issuer and the rule has a condition that requires that there is no chain of delegation in the access request environment.

An administrative level permission will contain an access rule, with target and condition, but also a condition with a delegation constraint. The condition will return true only if there is, in the environment of the access request, a chain of delegation which satisfies the delegation constraint. Administrative level permissions also have issuers.

When the Policy Decision Point sees an XACML policy with an issuer it will automatically add an implicit obligation to that policy. The added obligation is to perform another access request to check that the policy was authorized by another policy. To construct the implied obligation the PDP takes the issuer of the policy and adds it to the eventual existing delegation chain from the current request environment. The new request is the same access request with the new chain of delegation in the environment. The new request can in turn lead to a new obligation, and so on.

The recursion ends in a specific trusted root issuer. Frank Siebenlist calls it “PDP”, while we have called it “root” in our previous work. A PDP is assumed to not accept any “root” issued policies except from special closely located, trusted policy information points that have been initialized by means of special procedures. As an alternative, there could be a global agorithm to derive the root for an access request from the service name. For instance the root authority could be a part of the service name.

Some trade-offs

If we understand Frank Siebenlist’s proposal correctly, he suggests that rules are combined ahead of the access time, so the recursive requests are not needed. Compared to our suggestion, this has the advantage that there is less work to be performed at access time. However Franks Siebenlist’s approach also has drawbacks. Combining rules requires that it is possible to calculate whether one condition expression is a restriction of another. As Tim Moses points out, this may be possible in some cases, but likely not possible in general. This would mean that we would have to give up some forms of conditions, thus XACML with delegations would not be as expressive as without. We choose to not compare conditions, but instead test the access request on all of the conditions, thus we can use any condition expression.

We have not seen constraints on delegation in any of the current work on adding delegation to XACML. Constrained delegation has at least one trade-off associated with it. We can check the delegation constraints either when a policy is added to the policy database, or at every access time. The semantics are not the same, and it is perhaps not obviously clear which sematics are “better”. Checking once is more efficient, but we think that checking the delegation constraints at access time is perhaps easier to understand for the user. Checking at access time means that if someone issues an administrative authorisation with a delegation constraint he can know that at any given time, an access cannot happen unless the constraints that were specified are still valid through the whole chain of delegation. Checking only once would mean that instead each person in the chain of delegation at some point of time satisfied the constraints in the administrative authorisation. Unpredictable distribution of policies means that the time when the check is done may be unpredictable. (Of course this discussion does not rule out optimizations such as caching.)

It is desirable that there are upper bounds on how complicated an access control decision check can be. We are currently considering a number of limitations to the model to give such bounds. This still remains work in progress and will be explored in the implementation experiments which we will perform in the near future. One example is to limit the maximum length of a delegation chain. Another example is to add an id, which will indicate the other rule which will give support to a given rule. In this way, when the PDP encounters an obligation implied from a policy issuer, it will know exact which rule to search for.

Furher Work

We are also interested in the administration of attributes. This currently falls outside the scope of XACML, but we will explore the possibilities to use XACML to indicate permissions to administer attributes. This will add additional complexity to requests since finding the attributes of users would result in additional XACML requests. Inherited attribute values presents another complication. These issues will also be explored by us in the implementation experiments.

Administration of access control needs to cover the removal of permissions in addition to the creation of them. In our earlier work we have used revocation to remove permissions, but we have not considered revocation of XACML rules yet.

In our earlier work we have not had any negative permissions. XACML supports negative permissions, so we need to consider their impact on our model. We have yet to considered how to handle the rule-combining algorithms of XACML.

Examples

The last page contains some early examples (in “pseudo-XACML”) on what we wish to do. They are for our own purposes only and do not represent any kind of proposed changes to the XACML language at this stage.

The first request is permitted by the second policy under the indicated obligation. The obligation leads to the second request, which is permitted by the first policy. In this case the first policy does not lead to another obligation. If it had not been issued by “root”, then there would have been a third request, with an environment with a delegation sequence that was two “steps” long.

References

[1] Tim Moses, XACML delegation use-cases, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[2] See discussion by Frank Siebenlist, Tim Moses, Anne Andersson and others on the XACML mailing list.
[3] Frank Siebenlist, Modeling Delegation of Rights in a simplified XACML with Haskell, http://www-unix.mcs.anl.gov/~franks/haskell/XacmlDelegationHaskell0.html
[4] Olav Bandmann, Mads Dam, and B. Sadighi Firozabadi. Constrained Delegations. In *proceedings of 2002 IEEE Symposium on Security and Privacy*, 2002
[5] B. Sadighi Firozabadi, M. Sergot, and O. Bandmann. Using Authority Certificates to Create Management Structures. In *proceedings of Security Protocols, 9th International Workshop, Cambridge, UK*, April 2001

```
<Policy>
  <Target>...</Target>
  <Issuer>root</Issuer>
  <Rule RuleId="Rule1" Effect="Permit">
    <Target>
      <Subjects><Subject>mirty@sics.se</Subject></Subjects>
      <Resources><AnyResource/></Resources>
      <Actions><AnyAction/></Actions>
    </Target>
    <Condition
      FunctionId="urn:sics:function:delegation-sequence-match">
      <AttributeValue
        DataType="urn:sics:data-type:delegation-constraint">
        <step><subject>babak@sics.se</subject></step>
      </AttributeValue>
      <EnvironmentAttributeDesignator
        AttributeId="urn:sics:names:environment:delegation-sequence"
        DataType="urn:sics:data-type:delegation-sequence"/>
    </Condition>
  </Rule>
</Policy>
```

```
<Policy>
  <Target>...</Target>
  <Issuer>babak@sics.se</Issuer>
  <Rule RuleId="Rule2" Effect="Permit">
    <Target>
      <Subjects><Subject>mirty@sics.se</Subject></Subjects>
      <Resources><AnyResource/></Resources>
      <Actions><AnyAction/></Actions>
    </Target>
    <Condition><!-->No delegation-sequence in environment<--></Condition>
  </Rule>

  <Obligations>   <!-- Implied, not part of actual policy -->
    <Obligation
      ObligationId="urn:sics:obligation:authorize-issuer"
      FulfillOn="Permit">
      <AttributeAssignment AttributeId="urn:sics:attribute:issuer"
       >babak@sics.se</AttributeAssignment>
    </Obligation>
  </Obligations>
</Policy>
```

```
<Request>
  <Subject>mirty@sics.se</Subject>
  <Resource>vault</Resource>
  <Action>open</Action>
</Request>
```

```
<Request>
  <Subject>mirty@sics.se</Subject>
  <Resource>vault</Resource>
  <Action>open</Action>
  <Environment>
    <Attribute
      AttributeId="urn:sics:names:environment:delegation-sequence"
      DataType="urn:sics:data-type:delegation-sequence">
      <AttributeValue>
        <step><subject>babak@sics.se</subject></step>
      </AttributeValue>
    </Attribute>
  </Environment>
</Request>
```