[OpenAz] Version 88 has been uploaded supporting PolicySets in Policy Builder

Rich.Levinson rich.levinson at oracle.com
Mon Apr 19 00:07:55 EDT 2010

- Previous message: [OpenAz] Callback in the middle of a policy decision evaluation
- Next message: [OpenAz] Understanding query and scope semantics/implementation
- **Messages sorted by:** [date] [thread] [subject] [author]

To OpenAz emaillist:

Version 88 has an enhanced policy builder module, which incorporates PolicySets. The main purpose of this capability is to support a version of query that we have been experimenting with.

Attached are some sample files including the new policybuilder:

* TestAzApiPolicySets.java - the sample policy builder
* TestAzApi-GeneratedPolicy.xml - the PolicySet generated by the above policy builder
* SampleRequestForAccess.xml - a request to access a resource
* SampleResponseToAccessRequest.xml - a response to the above request
* SampleRequestForQuery.xml - a request to query what resources can be accessed by the User1
* SampleResponseToQueryRequest.xml - a response to the above query request

This approach realizes a notion that has been kicking around for a while, that regular expressions can be used for hierarchical resources, and that within this context, all that's really needed for a query is to return the list of regular expressions that are associated with the policies applicable to the Subject of the Request.

To step back for a moment, one must consider what is meant by querying for a scope of resources that a user has access to. There are a few assumptions that need to be made to get this notion in tractable form:

* What "type" of resources are within the scope? This question implies that the scope should specify the "type" of resources for which the query is being issued. Also, it raises the question of "what if the desired query is for any type of resource?".
* To keep things manageable, the assumption is made that the resources are hierarchical and there can be any number of types of hierarchical resources.
* The implication of the above assumption, is that the response to the query should provide for identifying the type of resource along with identifying the resource itself.
* How is the PDP supposed to know what resources the user has access to? All the PDP has access to, in general, is the Request and the Policies. Given that this is generally sufficient information to determine if the user has access to any specific resource, apparently the PDP does not have to know about the existence of specific resources, since the policies do not generally contain information about specific resources.
* Therefore, the conclusion is that all the PDP can possibly do, is return the policy information that describes the resources the user has access to, and leave it to the caller to use that information to construct any lists of concrete resources that the user has access to.
* With hierarchical resources and regular expressions, the solution to the above situation is straight-forward: namely the query result will be the list of regular expressions that are applicable to the user and the types of resources that are being requested for that user.

Given the above analysis, the only thing left is how to construct XACML policies that will give the desired results.

The attached policy TestAzApi-GeneratedPolicy.xml shows the technique that was used:

* Basically the structure is shown in the PolicySet in the 2nd half of the attached policy: PolicySetId="*QueryTest-PolicySet*"
* The structure of this PolicySet is as follows:
  - It contains one child PolicySet for each User. In this case there is only one user in the example and the PolicySet is PolicySetId="*QueryTest-User1-PolicySet*".
  - This PolicySet contains one child PolicySet for each "permission" granted to the user. In this example 2 "Permissions" were granted to the user in the following PolicySets:
    + PolicySetId="*QueryTest-User1-Rule1-PolicySet*"
    + PolicySetId="*QueryTest-User1-Rule2-PolicySet*"
  - Each of the above PolicySets contain 2 "Policy"s
    + PolicySetId="*QueryTest-User1-Rule1-PolicySet*"
      # PolicyId="*Policy-User1-Rule1*" Target = "http://www.example.com/A/B/.*"
      # PolicyId="*Policy-User1-Rule1-Query*" Target = "/-"
    + PolicySetId="*QueryTest-User1-Rule2-PolicySet*"
      # PolicyId="*Policy-User1-Rule2*" Target = "http://www.example.com/A/D/.*"
      # PolicyId="*Policy-User1-Rule2-Query*" Target = "/-"

The basic technique is that for each user there is a PolicySet for each "Rule" that applies to that user. So, in the above case there are 2 rules that apply to User1:

* http://www.example.com/A/B/.*
* http://www.example.com/A/D/.*

i.e. that user can access anything under /A/B and /A/D, but nothing else. The regular expression construct ".*" matches any number of any characters that follow the root part of the expression.
Basically the first Policy in each pair of Policys specifies the rule to apply to the user.

The way the query operates is that rather than specifying a specific resource, the query specifies the specific string "/-" as the requested resource. Since all rules for the user have a Policy with this resource identifier, namely the 2nd Policy in the pair, then every policyset containing a rule for the user will be "applicable" to the request.

So, the final piece of the puzzle is that the "query" Policy of the pair includes an obligation that contains the regular expression that is used by the compainion Policy in the pair. So, repeating the above structure with the relevant obligations added, we have:

* Each of the above PolicySets contain 2 "Policy"s
  - PolicySetId="*QueryTest-User1-Rule1-PolicySet*"
    + PolicyId="*Policy-User1-Rule1*" Target = "http://www.example.com/A/B/.*"
    + PolicyId="*Policy-User1-Rule1-Query*" Target = "/-" Obligation AttributeAssignment = "http://www.example.com/A/B/.*"
  - PolicySetId="*QueryTest-User1-Rule2-PolicySet*"
    + PolicyId="*Policy-User1-Rule2*" Target = "http://www.example.com/A/D/.*"
    + PolicyId="*Policy-User1-Rule2-Query*" Target = "/-" Obligation AttributeAssignment = "http://www.example.com/A/D/.*"

If one looks at the SampleRequestForQuery and the SampleResponseToQueryRequest, one will see these Obligations returned.

For the purposes of explaining the basic structure, I have left out a number of useful details, but one should be able to figure most of those out by looking at the PolicySets and the associated code for generating those PolicySets and the sample request and response, all of which are attached.

Note that the everything was generated using SunXacml and TestStyles.java test program, which has added a 4th test that uses the PepApi to generate the requests and handle the responses that are attached above.

One final note on the policy builder. It is obvious that the code is getting bloated with lots of repetitious constructs. Being that this is an open source project, this should be looked at as an opportunity for some creative energy to be applied to further consolidate and build out the basic patterns that have been established, so that the policy tool can:

1. Accept the minimum of input necessary to describe the policies
2. Display the policies in some useful form to be able to easily recognize the structure of the overall policy.

Also, for those who follow the xacml email lists, there has been some discussion on this topic of query in recent days:

http://lists.oasis-open.org/archives/xacml-users/201004/maillist.html
http://lists.oasis-open.org/archives/xacml-users/201004/msg00003.html

It is my assumption that the structure described above in this email and the implementation of it in Version 88 of OpenAz provides a potential solution to the problems described on the mailing lists and to query in general. Of course the current solution is intended only to demonstrate feasibility, as it is expected that in order to scale that optimizations will need to be made.

Thanks,
Rich

-------------- next part --------------
An HTML attachment was scrubbed...
URL: http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0001.html
-------------- next part --------------
An embedded and charset-unspecified text was scrubbed...
Name: TestAzApiPolicySets.java
Url: http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0001.pl
-------------- next part --------------
A non-text attachment was scrubbed...
Name: TestAzApi-GeneratedPolicy.xml
Type: text/xml
Size: 26291 bytes
Desc: not available
Url : http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0005.xml
-------------- next part --------------
A non-text attachment was scrubbed...
Name: SampleResponseToQueryRequest.xml
Type: text/xml
Size: 1547 bytes
Desc: not available
Url : http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0006.xml
-------------- next part --------------
A non-text attachment was scrubbed...
Name: SampleRequestForAccess.xml
Type: text/xml
Size: 1559 bytes
Desc: not available
Url : http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0007.xml
-------------- next part --------------
A non-text attachment was scrubbed...
Name: SampleRequestForQuery.xml
Type: text/xml
Size: 1534 bytes
Desc: not available
Url : http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0008.xml

```
-------------- next part --------------
A non-text attachment was scrubbed...
Name: SampleResponseToAccessRequest.xml
Type: text/xml
Size: 728 bytes
Desc: not available
Url : http://lists.openliberty.org/pipermail/openaz/attachments/20100419/b8e477e1/attachment-0009.xml
```

- Previous message: [OpenAz] Callback in the middle of a policy decision evaluation
- Next message: [OpenAz] Understanding query and scope semantics/implementation
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the OpenAz mailing list