Hecate, Managing Authorization with RESTful XML

Sebastian Graf, Vyacheslav Zhouldev, Lukas Lewandowski, and Marcel Waldvogel

Distributed Systems Group
University of Konstanz
&
School of Engineering and Science
Jacobs University
Problem Statement

How can authorization on RESTful resources can be performed under the following constraints?

1. Variability of resources must be kept!

2. Different permissions must be offered for the same resource!

3. The extensibility of the framework must be ensured!
Schema of Architecture

Requests & Responses

PXD

Authorization Framework

user model

Hecate

Resources

firstname : "Peter"
lastname : "Paul"

"lecture": {
"start": "April 4th",
"end": "June, 31st,
"topic": "Web Apps"
}

<html>
</html>

Sunday, March 27, 2011
User Model

- Simple mapping of users to rules
- Represented as a simple table

<table>
<thead>
<tr>
<th>user-id</th>
<th>rule-ids</th>
</tr>
</thead>
<tbody>
<tr>
<td>john.doe</td>
<td>13</td>
</tr>
<tr>
<td>jane.doe</td>
<td>12 13</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Permission XML Model (PXD)

- XML-based definition of operations mapped to resources
- Representing rules and their mapping to
  - HTTP-functionalities
  - Resources
  - (Optional resource aware filters)
4 Properties of PXD

1. One URI can be guarded by multiple rules
2. Each rule maps to one HTTP-Verb
3. Resource-awareness is possible
4. The representation of the resource is independent of the PXD
PXD Example

```xml
<resources>
  <rule id="12" perm_id="22"
       uri="http://house/floor/4"/>
  <rule id="13" perm_id="23"
       uri="http://house/floor/4"/>
...
  <data uri="http://house/floor/4">
    <content>
      /house/floor[@id=4]
    </content>
  </data>
...
  <filter id="43">
    <link>
      /house/floor[@id=4]//lamps
    </link>
  </filter>
  <perm id="22" filter_id="43"
       verb="get"/>
  <perm id="23" verb="get"/>
</resources>
```

Diagram:
- Data node with attributes `@uri` and `@perm_id`
- Filter node with attributes `@filter_id` and `@id`
- Rule node with attributes `@rule_id` and `@id`
- User node with attributes `@user_id` and `@rule_ids`

Example table:
- John Doe: 13
- Jane Doe: 12 13
- …
Workflow of Authorization

1. Receiving Request
2. Get IDs of user model related to credentials
3. Get rules matching the URI, the REST verb and the IDs
4. IDs found?
   - Yes: Applying filter on either request or result and returning result
   - No: Nodes found?
5. Nodes found?
   - Yes: Checking optional filter for matching permission
   - No: Returning 403 (forbidden)
6. Returning 403 (forbidden)
7. Forwarding request and returning result
8. Filter available
Resource-aware filtering

- Optional
- Bound to the requesting resource
  - Kind
  - Content
  - Filter possibilities
- Flexible mapping to HTTP-Verbs
Example

```
<house>
  <floor id="1">
    <room>
      ...
      <lamps>
        <lamp status="OFF" id="1.1"/>
        <lamp status="OFF" id="1.2"/>
      </lamps>
    </room>
  </floor>
  <floor id="4">
    <room>
      ...
      <lamps>
        <lamp status="OFF" id="4.1"/>
        <lamp status="OFF" id="4.2"/>
      </lamps>
    </room>
  </floor>
</house>

(resources>
  <rule id="12" perm_id="22"
    uri="http://house/floor/4"/>
  <rule id="13" perm_id="23"
    uri="http://house/floor/4"/>
  ...
  <data uri="http://house/floor/4">
    <content>
      /house/floor[@id=4]
    </content>
  </data>
  ...
  <filter id="43">
    <link>
      /house/floor[@id=4]//lamps
    </link>
  </filter>
  <perm id="22" filter_id="43"
    verb="get"/>
  <perm id="23" verb="get"/>
</resources>
```
Example

```xml
<house>
  <floor id="1">
    <room>
      ...
      <lamps>
        <lamp status="OFF" id="1.1"/>
        <lamp status="OFF" id="1.2"/>
      </lamps>
    </room>
  </floor>
...
  <floor id="4">
    <room>
      ...
      <lamps>
        <lamp status="OFF" id="4.1"/>
        <lamp status="OFF" id="4.2"/>
      </lamps>
    </room>
  </floor>
</house>

<resources>
  <rule id="12" perm_id="22">
    <uri>http://house/floor/4/</uri>
  </rule>
  <rule id="13" perm_id="23">
    <uri>http://house/floor/4/</uri>
  </rule>
...
  <data uri="http://house/floor/4">"/>
    <content>
      /house/floor[@id=4]
    </content>
  </data>
...
  <filter id="43">
    <link>
      /house/floor[@id=4]//lamps
    </link>
  </filter>
  <perm id="22" filter_id="43">
    verb="get"/>
  <perm id="23" verb="get"/>
</resources>
```
Read Requests

GET john.doe:secretpass@http://house/floor4

HTTP-REQUEST

DATA

PERMITTED DATA

http://house/ RESOURCE /house[./floor/@id=4][/lamps/*]

XPath RESPONSE
Write Requests

Concerns about authorized writes:

- Atomicity of modifications
- Visibility of data
- ...

Clear definition of constraints going along with modification requests:

- Weakening Authorization?
- Persistent Views?
Write Requests

Concerns about authorized writes:

- Atomicity of modifications
- Visibility of data
- ...

Clear definition of constraints going along with modification requests:

- Weakening Authorization?
- Persistent Views? (e.g. VDocs in XML)
VDocs

- (Persistent / In-Memory) views on XML
- Wrapping XQueries
- On-the-fly annotating of elements

Consists out of

- VDoc processor
- VDoc specification

In our context:

On-the-fly annotating of permissions to verify valid permissions on intermediate result
VDoc-Example

```xml
<house>
  <floor id="1">
    <room>
      ...
    </room>
    <lamps>
      <lamp status="ON" id="1323412"/>
      <lamp status="ON" id="5456"/>
    </lamps>
  </floor>
  ...
  <floor id="4">
    <room>
      ...
    </room>
    <lamps>
      <lamp vdoc:uri="..."
        vdoc:xpath="/house[1]/floor[4]/room[1]/lamp[1]"
        status="ON" id="3443"/>
      <lamp vdoc:uri="..."
        vdoc:xpath="/house[1]/floor[4]/room[1]/lamp[2]"
        status="ON" id="5456"/>
    </lamps>
  </floor>
</house>
```
Write Requests

HTTP-REQUEST

POST john.doe:secretpass@http://house/floor4

Body

for $status in /house//lamp/@status
return replace value of node $status with "ON"

mod.VDoc

VDoc

 Permission-XPath

VDoc Spec

only marked Nodes modified

Yes

No

Returning 403 (forbidden)

Apply Changes
Summary

- Flexible authorization framework fitting the extensibility of HTTP and REST
- Independence from resources
- Optional extension of authorization workflow with the help of in-depth knowledge
- Implementation in JAX-RX (Treetank) and TntBase
Thanks for your attention

Any Questions now?

(or later: sebastian.graf@uni-konstanz.de)