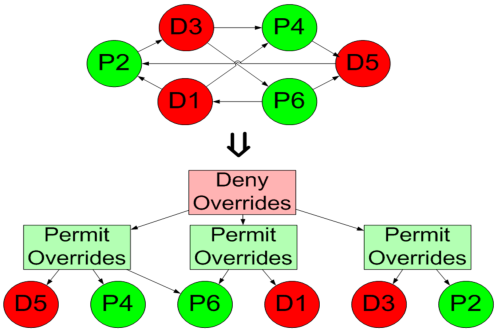
Stefan Oberholzer

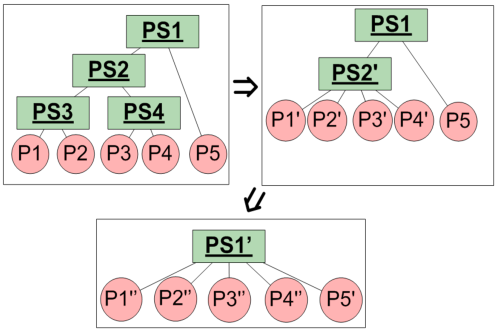
Graduate Candidate	Stefan Oberholzer
Examiners	Prof. Oliver Augenstein, Prof. Dr. Josef M. Joller
Co-Examiner	Dr. Günter Karjoth, IBM Forschungslabor Zürich, Zürich, ZH
Master Research Unit	Software and Systems

Optimizing XACML Policies

Conflict Detection, Transformation and Minimization



Creating a policy set out of policies with precedence definition



Flattening an existing policy tree

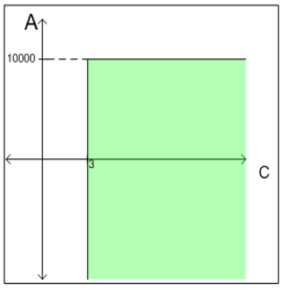
Introduction: Companies and other organizations are making more and more use of XACML- based access control solutions. Administrators of XACML access control policies are facing the problem of building an efficient policy set that resolves conflicts between policies as intended.

Objective: The following tasks on policies are defined in this thesis

- Define parallel algorithms to detect conflicts between policies.
- Define parallel algorithms to optimize policy sets
- This work focuses on XACML 3. As XACML 2 is in use in most companies and organizations a description will be given of how the proposed algorithms and solutions can also be applied to it.

Result: Two parallel algorithms to detect conflicts between policies are proposed. The first allocates the extreme points of the policies to the processes. Each one detects conflicts and forwards the result to the neighboring processes if a policy cannot be completely resolved. The other algorithm is optimized for the case where a large number of policy groups exist only in conflict with each other. In the case of policy set optimization an algorithm to minimize targets is proposed. In addition to this, operations on a policy set are defined. This includes adding a policy to resolve the conflicts as intended and transforming the policy tree to reduce the number of levels, or to add additional levels in the policy tree defined by the policy set. To test the correctness of these operations an existing mapping to description logic has been extended with the algorithms new to XACML 3. The proposed algorithms support an administrator in creating a policy tree by optimizing targets, detecting conflicts and creating a policy tree out of a set of rules. The operations optimize the evaluation performance of a policy set.

```
<Policy PolicyId="P1"
  RuleCombiningAlgId="[…]permit-overrides">
  <Description>
    Example Policy matching to all Requests. Combining algorithm is
Permit-Overrides.
  </Description>
  <Target/>
  <Rule RuleId="ExampleRule" Effect="Permit">
    <Description>
      Contained rule with Permit effect. Matching to all Requests
containing role-id with value greater than 2.
    </Description>
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="[…]integer-less-than">
            <AttributeValue DataType="[…]integer">10000</AttributeValue>
            <AttributeDesignator MustBePresent="true"
              category="[…]resource"
              AttributeId="A" DataType="[…]integer"/>
        </Match>
        <Match MatchId="[…]integer-greater-than">
            <AttributeValue DataType="[…]integer">3</AttributeValue>
            <AttributeDesignator MustBePresent="false"
              category="[…]subject"
              AttributeId="C" DataType="[…]integer"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
  </Rule>
</Policy>
```



$$Permit\text{-}E_{P1} \coloneqq \exists\, A.<_{10000} \sqcap \exists\, C.>_{3}$$
$$Deny\text{-}E_{P1} \coloneqq \bot$$
$$IndetP\text{-}E_{P1} \coloneqq \forall\, A.\bot$$
$$IndetD\text{-}E_{P1} \coloneqq \bot$$
$$IndetDP\text{-}E_{P1} \coloneqq \bot$$

XACML policy in XML, description logic and graphical description