Overriding access control in XACML

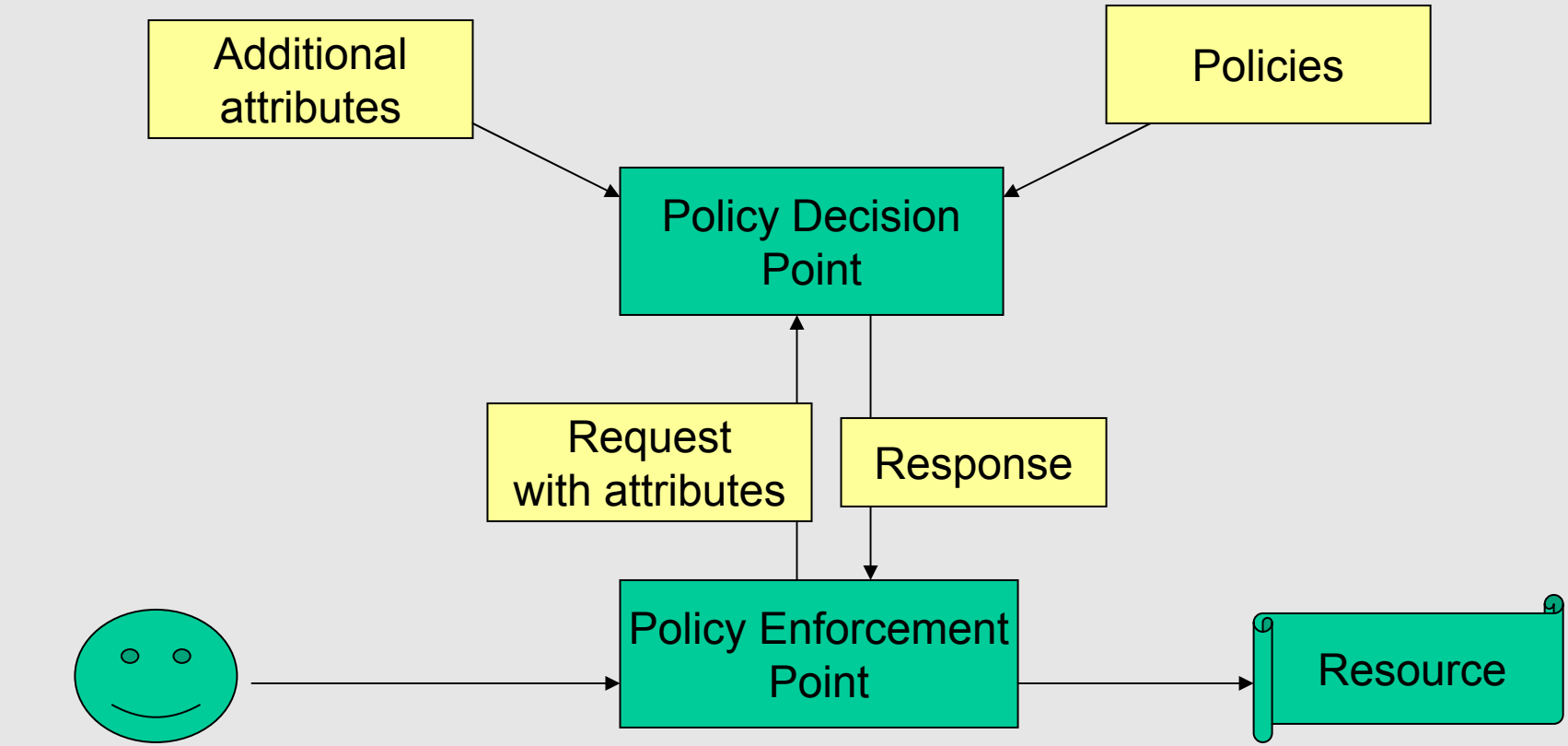
Ja'far Alqatawna
Erik Rissanen
Babak Sadighi

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

XACML, eXtensible Access Control Markup Language

- An XML-based access control policy language
  - This work based on the current version 2.0
- Information about an attempted access is described in terms of attributes of the Subject, Resource, Action and Environment
- Policies are functional expressions based on the attributes
- Output is a Permit/Deny/NotApplicable decision
  - (Or ”Inderminate” which indicates an error)

XACML architecture



Policy Combining in XACML

- Policies can be collected in PolicySets
- All policies are evaluated separately
  - Each will say Permit/Deny/NotApplicable
- A Policy Combining Algorithm is used to resolve conflicts
  - ”permit overrides”, ”deny overrides”, ”first applicable”, etc

Obligations in XACML

- A policy may contain ”Obligations”
- An obligation consists of an identifier and optional parameter values
- An obligation is an additional action which the policy enforcement point has to implement
  - The identifier defines the semantics of the obligation

Issues with obligations

- The treatment of obligations is quite simple
  - They are simply collected into a set from the applicable policies
- There is no mechanism for resolving conflicts between obligations
  - For instance ”log in detail” vs ”protect privacy”
- This paper contributes a conflict resolution mechanism for obligations

Previous work on generalization

- Categorization by Michiharu Kudo
  - Aim to understand what use cases there are for more general treatment of obligations in XACML
  - Atomic, Sequential, Asynchronous, Supplemental, Data-processing
- Bill Parducci has described the categories in terms of parameters
  - Exclusive, Timing, Sequence

Access control override

- We have used access control override as our use case
- Consider the policy:
  - A doctor may read the records of any patient of which he is the primary physician
  - A doctor may read the records of any patient whose life is at threat
- The second rule cannot be implemented on a computer
  - A computer cannot know if life is at threat or not
- But we don't want to close up the system either
  - That could be fatal, literally…
- Solution: let the doctor make the decision, but audit extra carefully to prevent abuse
  - Need to mark certain rules for strict audit
  - Better than fully open system: less logs to look at

Implementing override in XACML

- Essentially we need to move beyond the Permit/Deny decision of XACML
  - Permit, Deny, Override
- Obligations could be a simple method to do this
  - Define an obligation which means that a warning message is displayed and a special audit log record is written
  - Three possible decions: Deny, Permit and Permit with override obligation

The problem using obligations

- How do we resolve the conflict in decision between a ”regular permit” and a ”permit with an override obligation”?
  - If both would apply to a request, we want the normal permit to have precedence
- XACML lacks this kind of conflict resolution

Solving with ordered combining

- One approach is to use a first applicable policy combining algorithm
- Just put the ”regular permits” first in the policy set
  - The override policies/rules will never be reached if a regular permit is applicable
- Problem: keeping policies in order may not be practical in a distributed administration case
  - This leads to a need of a global view of the policies and a risk that someone messes up the order

Solving with a custom policy combining algorithm

- It is (at least practically) possible to write a policy combining algorithm which looks at the obligations as well, in addition to the Permit/Deny decisions
- However, this is a bit of a kludge, and it would be better to have a more explicit, standardized solution

What we did

- First combine effects from policies
  - (Essentially the regular policy combining algorithms)
- After this, a number of obligation combining algorithms may be called
  - Each obligation combining algorithm recognizes particular types of obligations, removes conflicts and passes the others to the other algorithms
- At the end, the remaining obligations are returned to the PEP

The effects combining algorithm

- Combines the policies into an aggregate decision and collects obligations into a list
  - (In contrast to a set in plain XACML)
- The output looks like this:
  - <Effect, [obl1, obl2, …, oblN]>
  - Effect is the combined decision (Permit/Deny)
  - obl1, …, oblN are sets of obligations from the policies (kept separate)

Obligation combining input

- An obligation combining algorithm takes as input:
  - <Effect, [obl1, obl2, …, oblN], Obls, WS>
  - Effect is the decision of the policy set
  - obl1, …, oblN are sets with the obligations to combine
  - Obls is a set of obligations from the policy set itself
  - WS is a working set of already combined obligations
    - (WS starts empty)

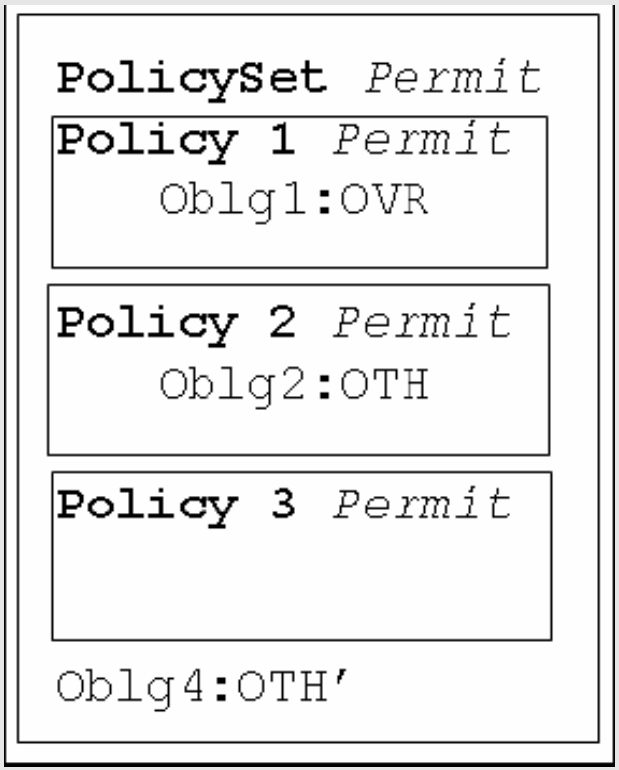
Obligation combining chaining

- The output of an obligation combining algorithm is of the same form as the input
- The algorithm is free to remove any obligations it recognizes from the list and ”Obls” set
  - After any conflicts have been resolved, the output of the algorithm is placed in the working set
- The input of the first obligation combining algorithm is the output of the effects comb alg
- The output of an obligation combining algorithm is the input of the next obligation combining alg

Schema changes

- A new element called <OblgCombAlg> as a child to the <PolicySet> element
- This element lists the obligation combining algorithms which should be applied

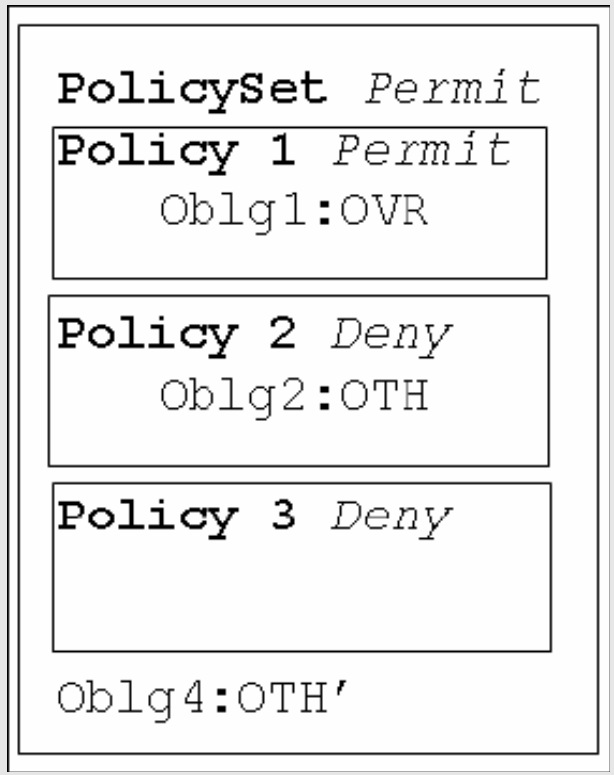
Example 1



- Combine effects (permit overrides) and collect obligations:
- <Permit, [[OVR],[OTH],[]],OTH',[]>
- Override-combining algorithm gives priority to policy without OVR obligation and removes the OVR obligation:
- <Permit, [[],[OTH],[]],OTH',[]>
- Any other obl comb algs could be called. (Not shown)
- The final result will not contain the OVR obligation

Example 2

- Combine effects (permit overrides) and collect obligations:
- <Permit, [[OVR]],OTH',[]>
- Override-combining algorithm does not find a policy without an OVR obligation, so it collects the OVR obligation:
- <Permit, [[]],OTH',[OVR]>
- Any other obl comb algs could be called. (Not shown)
- The final result will contain the OVR obligation



Issues with this solution

- Bill Parducci's critique
  - Requires new code in the PDP for new types of obligations, which is not practical
- Does not take into account all use cases from Michiharu Kudos work on categories
  - For instance order or timing may be significant

Recent work in the XACML TC

- Recent work (after paper submission) by Bill Parducci and Erik Rissanen
- Build on top of this paper and the work by Bill Parducci and Michiharu Kudo
  - Basic idea is still an obligation combining algorithm which recognizes particular obligations
    - But it is now called ”Obligation family”
  - Family templates based on ideas from Bill Parducci
    - The composite obligations can be defined in a policy, rather than being part of the algorithm definition
    - Parameters of families affect behavior
      - Inspired by use cases by Michiharu Kudo
    - Takes order of obligations into account

Conclusions

- We can solve the override use case
- We have provided a first simple approach to resolve conflicts between obligations in XACML
- Further work will allow more complex use cases and easier implementation

Questions?