The Ponder
Policy Specification Language

Markus Lorch
Motivation

♦ (security) management for large scale IT systems challenging topic

- number of components in enterprise-wide networks and distributed systems is high

- dynamic computing paradigms like active networks, mobile agents increase security concerns

- changes to system behavior should be possible on the fly and not require code changes
PONDER Policies

♦ Policies are rules that govern the choices in the behavior of a system
♦ Business agreements (e.g., service agreements) drive the definition of policies. Top level policies are typically abstract and must be refined for implementation in a specific system
♦ PONDER provides a language for the encoding of - management and - security (access control) policies
♦ The PONDER language is declarative and object oriented
Requirements for a Policy Language

♦ Support for access control, access right delegation and management activity
♦ Structuring techniques to improve management / scalability of policies
♦ Composite policies to combine/group security and management policies
♦ Able to analyse policies for conflicts and inconsistencies
♦ Extensibility
♦ Comprehensible language
PONDER policies are implementation independent and need to be compiled (mapped/translated) for a specific system.

Implementation examples for PONDER authorization policies
- mapping into Java access control policies
- mapping into Windows 2000 security templates and firewall rules
- mapping into Linux kernel access controls
Structuring Techniques

♦ A PONDER policy consists of a single rule

♦ PONDER policies can be declared directly or via the definition of parameterized policy types. (Reuse of declarations similar to template library)

♦ PONDER supports inheritance for extensibility
Structuring Techniques II

♦ Role-based access control allows for the grouping of individuals to improve scalability in large systems

♦ PONDER uses the organizing principle of “Domains” and “Sub-Domains”, which introduce a hierarchical grouping and naming scheme (similar to a directory structure)
PONDER Policy Types
Authorization Policies

♦ Authorization Policies
Can be positive or negative (what subjects may do, or what subjects may not do)

Example:

```c
inst auth+ switchPolicyOperators {
    subject /NetworkAdmin;
    target <PolicyT> /Nregion/switches;
    action load(), remove(), enable(), disable();
}
```
PONDER Policy Types II
Authorization Policies

♦ Information Filtering Policies
  Allow the filtering of parameters. E.g., to allow different “views” – some users may get more info back than others, but for all users the same rules were evaluated (no duplication of operations necessary)

♦ Delegation Policies
  Enable the specification of authority to grant rights to others. (grantee, subject, target, action, when, valid)
PONDER Policy Types II
Authorization Policies

♦ Refrain Policies
Define actions that subjects must not perform. Difference to negative access control policy is that these policies are enforced by subject (not by target). Targets are not trusted to enforce this policy. E.g. as target is not interested in protection from subject.
PONDER Policy Types III
Management Policies

♦ Obligation Policy
Defines event-based actions that must be performed. E.g. logging of unsuccessful login attempts (action) after three attempts (event).
PONDER Policy Types IV
Composite Policies

♦ Provide ability to group/structure policies following organizational structure

♦ Definition of roles (RBAC)
  - semantic grouping of policies with a common subject (aka role-definition, the subject is the role name, members of a role are defined elsewhere, aka role-allocation)
  - roles are thus a set of authorization, obligation, refrain and delegation policies

♦ Definition of Relationships
  define grouping of roles (subject and target of definition can be roles, defining their relationship)
What’s missing?

1. No model on how policies are introduced and applied in a system, this raises questions:
   - Ponder does not dictate how policies have to be processed / a decision reached (interoperability between different Ponder-based systems is questioned)
   - Several examples require state to be kept (e.g. event-based policies may need to count events)
   - What tells the compiler system what state to keep, and for how long, etc…?
What’s missing? II

♦ Policy / Rule combination
  How (e.g., in what order) are policies processed and how is the output combined

♦ Distributed policy authority, where is stated what policy was issued by whom, seems to rely on trusted repository for which it can enforce access

♦ How are attributes that are set in policy conveyed back to enforcement point