draft-ietf-i2nsf-capability-00 Development Plans

L. Xia, J. Strassner, C. Basile, D. Lopez

I2NSF Meeting,
Singapore,
November 14th, 2017

Introduction: the Context

- **Policy Enforcement Defined by *Capabilities***
  - Capability: the functions that an NSFs provides, *independent* of the customer and provider interfaces
    - An abstraction with well-defined semantics
    - Flexibility to represent functionality that can be either vendor-dependent or -independent
- **This Draft**
  - Defines the concept of NSF Capabilities and their use
    - Information model – characteristics and behavior in a protocol-, platform-, and vendor-independent manner
    - Info model defines a common lexicon for multiple data models
    - Capability Algebra – ensure that actions of different Policy Rules do not conflict with each other

Policy Rule – Capability Duality

- **Policy Rules Describe, Define, and Manage Capabilities**
  - *Policy Rules can be used to govern definition, configuration, monitoring, visibility, and usage of Capabilities*
  - *For example, Policy Rules can define:*
    - What is or is not a Capability
    - What Capabilities can be exposed to which consumers
    - Which OAM data is exposed to which consumers
- **Capabilities Define Reusable Functionality that is Manipulated by Policy Rules**
  - *Capabilities abstract the functionality of network elements into reusable objects that are used as building blocks to provide security features*
  - *Capabilities can be combined to provide more powerful features that are made selectively available to consumers (via Policies)*
  - *Capabilities enable security protection to be customized to suit the needs of the applications using it in a given context without relying on specific technologies or even vendors*

Key Abstractions

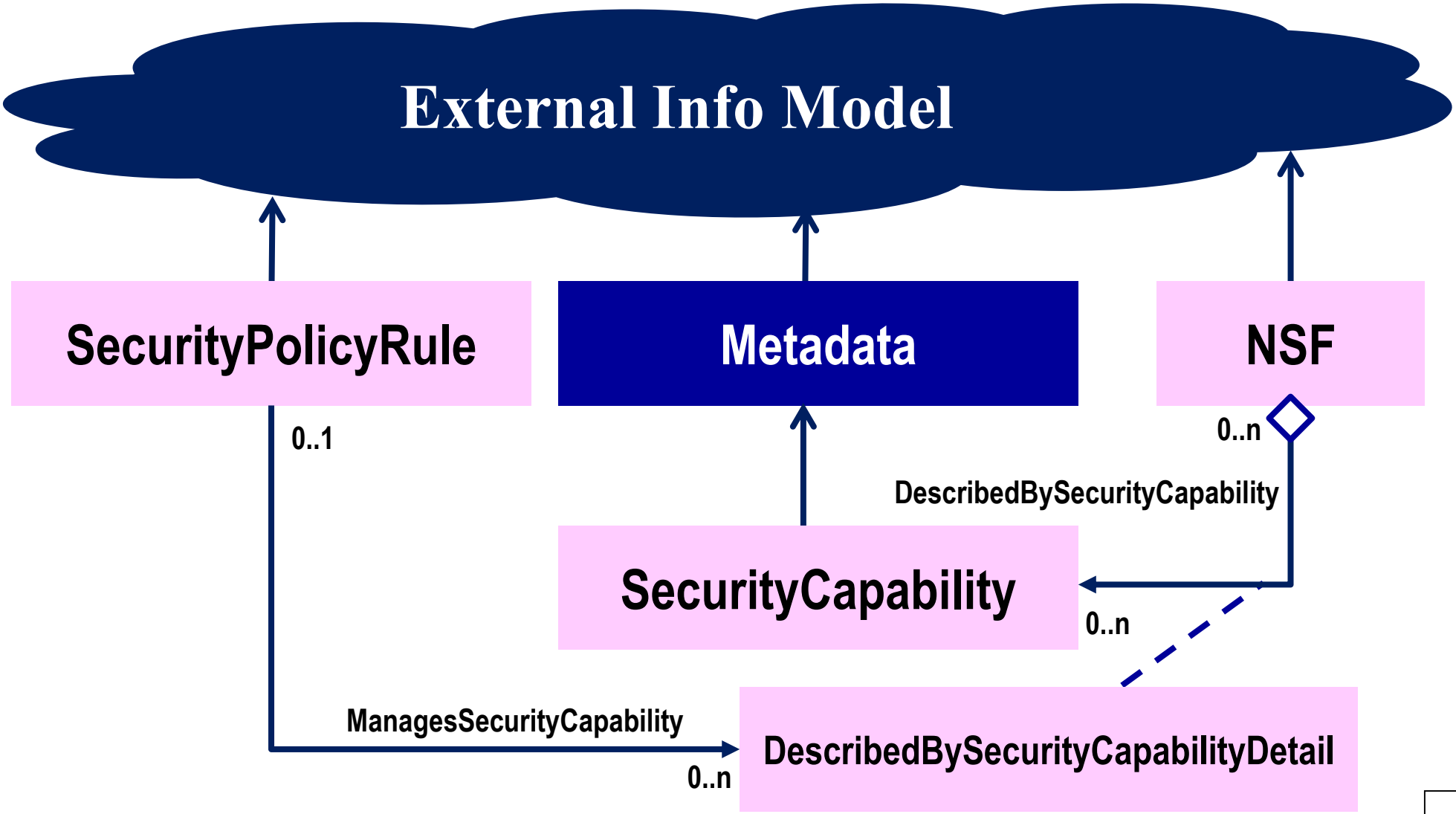
- **Security is independent of physical vs. virtual packaging**
- **Security is described by one or more Capabilities**
- **Policies define how to manage Capabilities**
- **Policies are defined in an object-oriented info model to maximize interoperability**

- **This enables**
  - *An infinite number of NSFs to be described and managed*
  - *An infinite number of Policy Rules to be defined to manage NSF behavior*
  - *Capabilities and Policy Rules to be reused as is, or for building more powerful Capabilities and Policies*

The ECA Policy Rule Model

- **The Current Model Uses ECA Policy Rules**
  - *Events:* significant occurrences the NSF is able to react to
  - *Conditions:* how the NSF decides which actions to apply
  - *Actions:* what operations to execute
  - *PolicyRule: a container that aggregates an Event, a Condition, and an Action (Boolean) clause*
- **Behavior**
  - Actions MAY execute if Event and Condition clauses BOTH evaluate to TRUE (both clauses are Boolean clauses)
  - Controlled by *resolution strategy* and *metadata*
    - Capability Algebra used to make resolution strategy decidable
  - *Default actions* MAY be specified

Conceptual Operation



Enhancements to the Capabilities I-D

- Improvements / extensions to consider for the next revision of this draft
  - Event clause / Condition clause representation
    - e.g., CNF vs. DNF for Boolean clauses
  - Event clause / Condition clause evaluation function
    - more complex expressions than simple Boolean expressions to be used
  - Action clause evaluation strategies
    - e.g., execute first action only, execute last action only, execute all actions, execute all actions until an action fails
  - More on metadata
    - authorship, time periods, (+ priorities)
    - more elaborate behavior description and specification

Switching to the Decorator Pattern

- **Categories and subcategories determined with sub-classing**
  - pros: intuitive, simple, easy to design
  - cons: not very elegant, requires non-trivial maintenance at every minor update, does not work well at run-time
- **The Decorator Pattern**
  - Defined in 1995 (!), used in java and windowing toolkits
  - much more expressive
  - reduces number of objects at runtime
  - provides dynamic behavior (composition) instead of fragile, inheritance-based behavior (which is static)

More Patterns

- **Define either an Appendix or a separate I-D to define and describe other patterns**
  - Patterns are templates that provide an abstract solution to a recurring situation that requires modeling
  - Large library of templates exist, but little use in networking (and especially security)
  - Next version of draft will restructure content to make maximal use of templates
  - Enables scalable solutions to be prototyped

Questions?

“Create like a god. Command like a king. Work like a slave”
- Constantin Brancusi