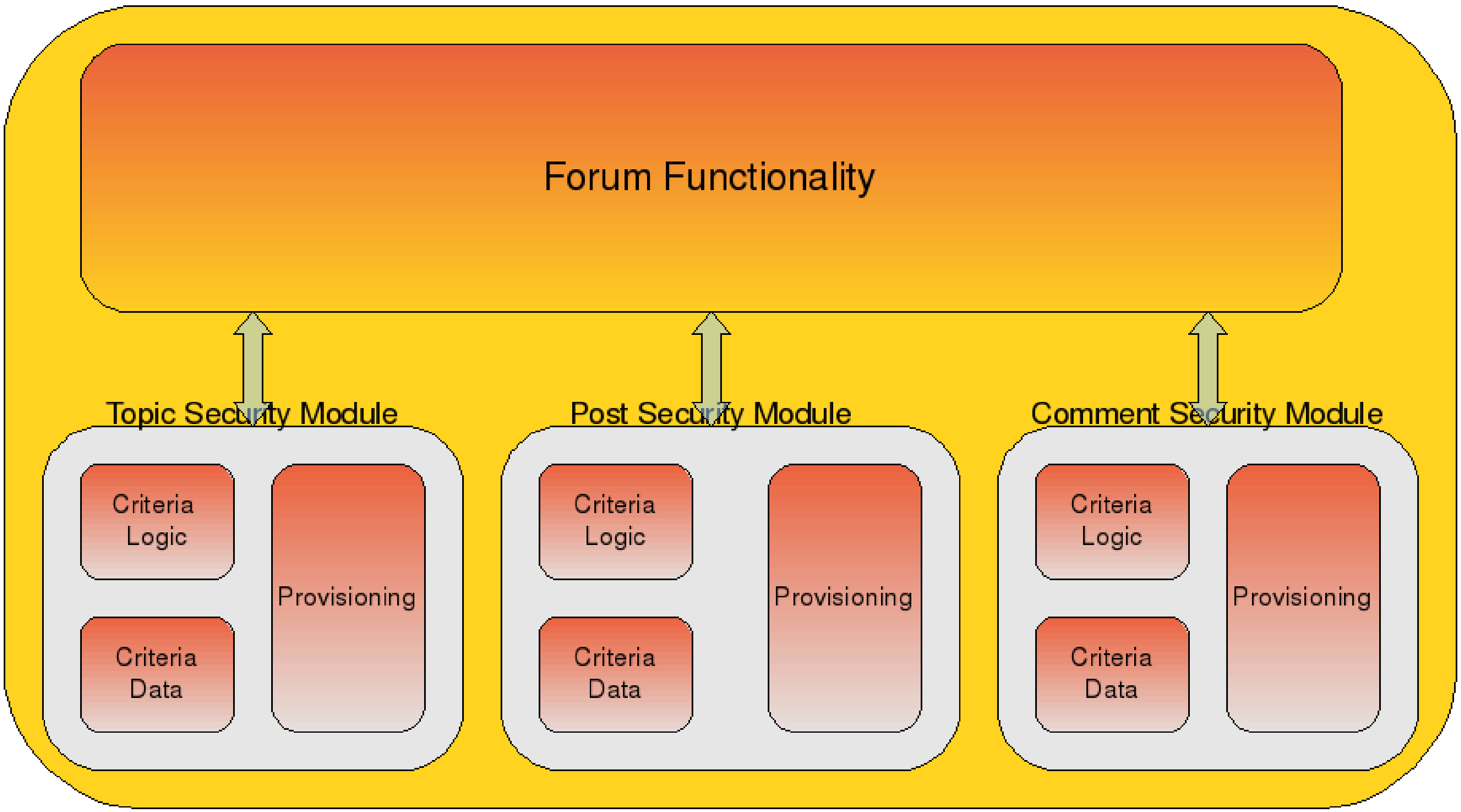
AuthZ - Authorization Framework

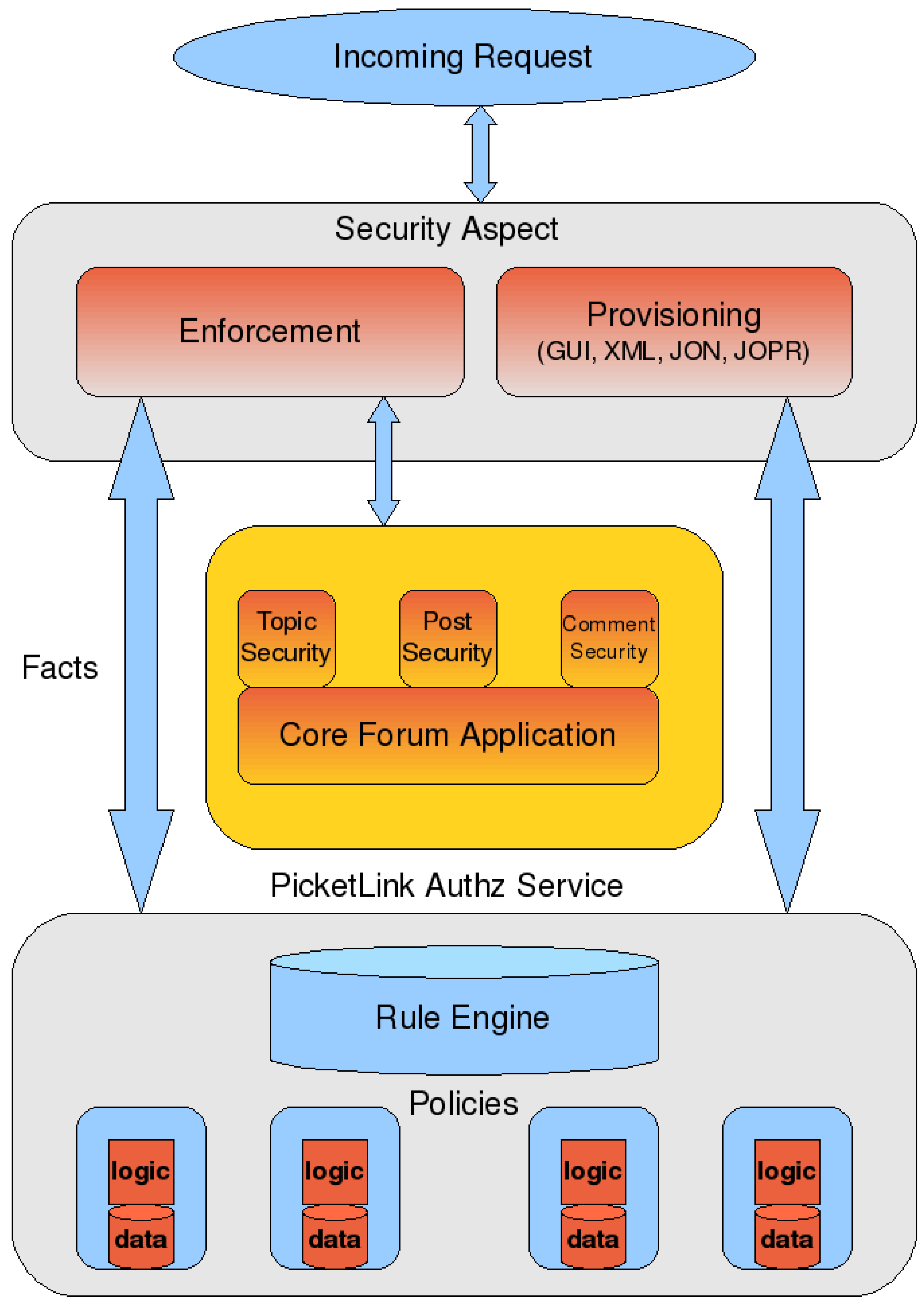
- *XACML-compliant* Rule-based authorization framework
- Externalizes Security (Logic + Data) from the core application into a Policy Engine
- Support for more robust security usecases in addition to Role-based Access Control (RBAC)
- Dynamic Policy Provisioning
- Consistent Developer Framework

Problem : What problem is being solved?

- Security codebase (logic+data) intermingled with application codebase
- Client code becomes brittle as its scattered throughout the codebase. This hampers quick evolution of security functionality
- Hard to customize security behavior and usecases
- An inconsistent approach across all modules. At best a mini-security module with its own classes, and data is created for each application module
- Provisioning tools don't have a common framework and infrastructure
- From persistence world: Same as JDBC/DAO vs ORM

Core Forum Application Codebase




Incoming Request
Security Aspect
Enforcement
Provisioning
(GUI, XML, JON, JOPR)
Facts
Topic Security
Post Security
Comment Security
Core Forum Application
PicketLink Authz Service
Rule Engine
Policies
logic
data
logic
data
logic
data
logic
data

Advantage: Security becomes a cross cutting concern

- Just like container-managed transactions, security becomes a container-managed cross cutting concern
- Application functionality can evolve without having to worry about security functionality in parallel
- Security functionality can evolve without affecting the codebase it protects
  - The concept of "Security Profiles" lets you swap different security behavior in and out. This is very useful for customization by developers, system integrators, and consultants
- Use of a "Fact" base approach decouples the traditional coupling between these two layers. From a code standpoint they don't know that the other one exists
  - It removes the application's need to designate points in the code "when/where" an authorization check is started
  - It does not have to specify "what action" needs protection
  - It does not have to issue any calls to the boolean logic that provides a "yes/no" decision

Advantage: Security Logic applied to arbitrary state

- Allows enabling robust access control behavior, without applying any added burden to core application codebase

- Some possible usecases for the same exact protected action
  - Allow access to this page if user is a member to this set of roles/groups (standard role-based access control)
  - Do not allow page access after 5 p.m.
  - Allow page access to all my friends from my social networking accounts
  - Allow page access only if the user is over 18 yrs old

- Since the security system is completely decoupled from the application, these security requirements/conditions can be customized for different customers. (same non-forked core codebase, different security behavior)

- The granularity of security enforcement can be customized
  - One installation only needs to protect the ”Page”
  - Another installation can go finer and protect ”Page”, ”Portlet Modes”, ”Window States”, and even the ”content” displayed inside the Portlet window

Advantage: Consistent Framework

- A component-oriented framework allows creation of policy-based provisioning tools
- Components encapsulate the (logic+data) aspect of a policy
- Components are re-usable across applications
- Developers use a common framework and runtime for developing the policy-management tools for their respective applications

Advantage: Policy Provisioning

- Provisioning tools can vary from xml based configuration, to integrated GUI based applications
- Provisioning is dynamic. All policy modifications are hot-deployed and activated across all applications without requiring any system restarts
- A consistent policy infrastructure and a common API even opens the door for integration with central monitoring tools like JOPR and JON

Concepts: Policy

- Policy is the central component of the framework. All authorization decisions are made by the ”Rule Engine” matching policies and executing the encapsulated ”Logic”
- It encapsulates externalized ”Logic” and ”Data” used to make an Access Control decison
- It answers the following question based on the incoming ”Facts”
  - Does this policy apply to this request
  - If it applies, do the incoming facts satisfy the conditions to be allowed access

Concepts: Policy

- The incoming ”Facts” provide the LHS/variable side of the ”Boolean” expression, while the ”Data” externalized within the Policy provide the RHS/criteria side of the expression
    - For instance, in a simple Role-based check,
        - Facts : ”Roles” of the currently authenticated user
        - Policy Data : ”Roles” specified to be allowed access

- The Policy along with its ”Logic” and ”Data” can be modified at runtime resulting in an entirely new security profile
    - An existing policy can have Role-based access control
    - New Logic related access based on ”the Age” of the user can be dynamically added

Concepts: Enforcement

- Enforcement consists of intercepting an incoming request, and evaluating whether it should be granted access to the resource
- It consists of populating an EnforcementContext with "Facts" about the request's environment
- Depending on the layer where its intercepted these "Facts" change
  - In the http layer facts look like,
    - "URL" of the resource
    - Request parameters
    - Request headers and cookies
    - Authenticated Identity and the correspoding "Roles/Memberships"
  - In a portlet layer facts look like,
    - "Name" of the Portlet being accessed
    - Phase such as "Render", "Action"
    - Portlet Mode
    - Authenticated Identity and the correspoding "Roles/Memberships", etc

Concepts: Enforcement

- The Enforcement Interceptor can be of various types:
  - In the Http layer, it could a Servlet Filter or a Tomcat Valve
  - In the Portlet layer, it could be a Portlet Filter
  - In Seam, it could be a Phase Listener
  - In pure POJO, it can be an AOP interceptor (my personal favorite)
- The EnforcementContext only provides the ″Facts″ (what I am trying to access, and the data in my environment). It does not contain any decision making logic. This ″logic″ is provided by the externalized ″Policy″ that will be matched with this particular request

Concepts: Provisioning

- The Provisioning phase is used by tool developers for managing security policies
- There are several types of tools possible depending on your application requirements
  - Simple XML based configuration
  - Dynamic GUI based Application
  - Plugins into central monitoring tools like JON and JOPR
- Whatever the tool is, they use the same common API and framework

Concepts: Provisioning

- The Provisioning framework uses Authz Component Specification compliant components.
- Each component encapsulates its ”Boolean Logic” (specified in the Drools language DRL), and ”Criteria Data” associated with the expression
- A set of components are orchestrated into an instance of a CompositionContext
- This instance of a CompositionContext is then processed by a PolicyComposer service. This service generates the system level XACML-compliant policy, and propagates it dynamically
- This allows a Developer to easy manage security policies without having to deal with the very complex low level details of XACML.

Questions!!