8.2 Policy

Intent

Isolate policy enforcement to a discrete component of an information system; ensure that policy enforcement activities are performed in the proper sequence.

Also Known As

Access Decision Function, Policy Decision Point [ISO / IEC 10181-3]

Motivation

Many systems (and components of systems) need to enforce policy. In such systems, the policy enforcement functions must be invoked, in the correct sequence, every time access is attempted to a resource which is subject to the policy.

If policy-enforcement code is intermingled with code which services resource requests, it may be difficult to verify that the policy enforcement functions are always invoked when necessary and that they are correctly implemented. It may therefore be desirable to isolate policy-enforcement code from other code in order to simplify verification of the correctness of the policy-enforcement code.

It may be desirable to use the same policy-enforcement code to protect more than one system component.

It may also be desirable to support changes in the code which makes policy decisions without requiring changes to code which enforces policy decisions.

Applicability

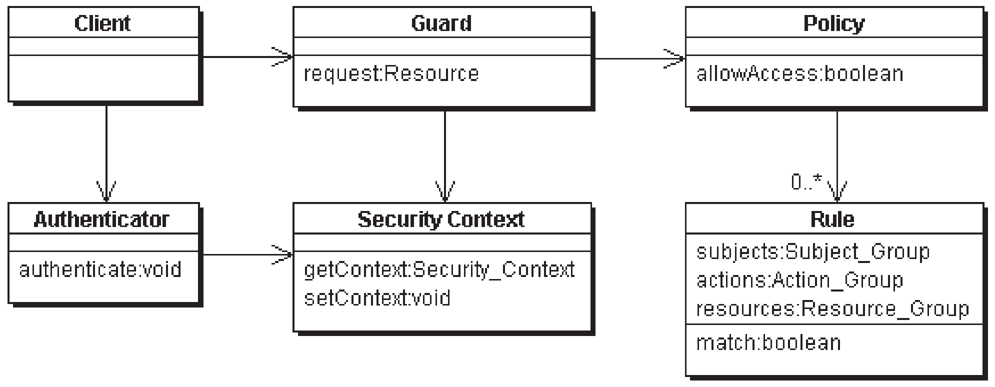
Use this pattern when:

- It is desirable to decouple the implementation of security policy from resource manager implementation.
- It is desirable to isolate policy-enforcement code to a minimum number of simple modules to simplify verification of correctness.
- It is desirable to isolate policy-enforcement code from policy decision evaluation code.

Do not use this pattern if:

- It is infeasible to make policy decisions outside the context of the resource manager which responds to requests.

Structure



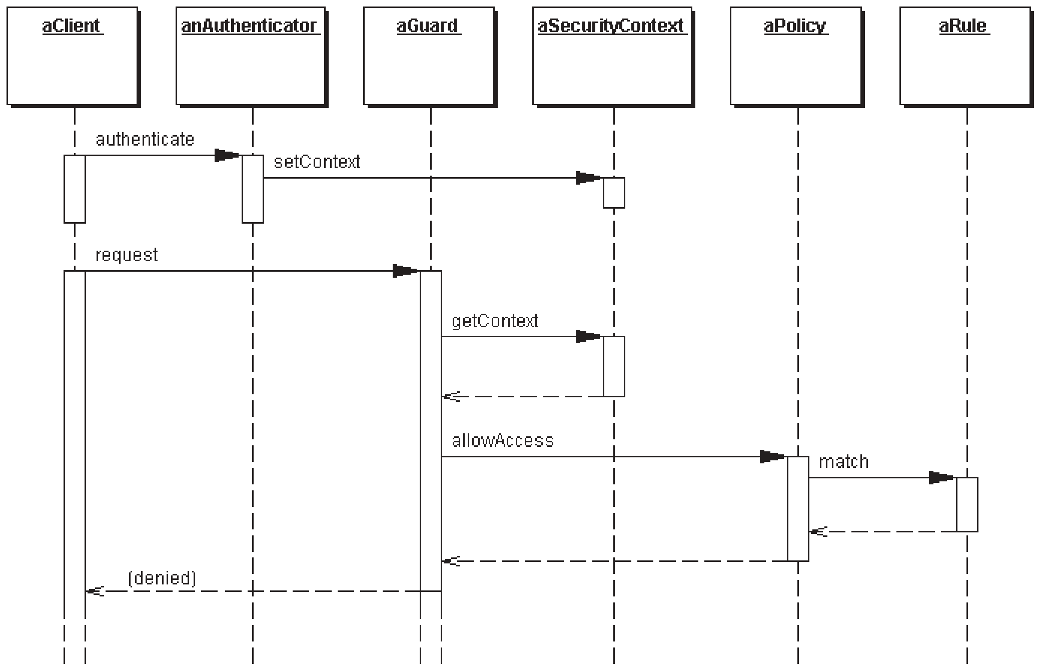
Participants

- Client
  - — Represents any subject governed by the system's policy.
  - — Gains access to resources by submitting requests to the Guard.
  - — May initiate user authentication if it is to operate on behalf of a human user.
- Authenticator
  - — Authenticates users.
  - — An optional component (the Security Context may be implicit or inherited).
- Guard (also known as Policy Enforcement Point (PEP))
  - — Collects client, request, target, and context attributes needed to make access control decisions.
  - — Requests access control decisions from the Policy.
  - — Rejects requests which are not permitted by the Policy.
  - — Sequences operations related to policy enforcement.
  - — May cache client identity and attribute information to optimize performance in cases where a single client submits multiple requests.
- Security Context
  - — Maintains credentials and security attributes for use in Policy decisions.
- Policy (also known as Policy Decision Point (PDP))
  - — Makes decisions to grant or deny access to resources based on client attributes, request attributes, target attributes, context attributes, and policy.
  - — Encapsulates a set of Rules determining which Clients can perform which operations on which Resource.

- Rule
  - — A component of the Policy expressing the permission for a specified set of Clients to perform a specified set of operations on a specified set of Resources.
  - — Structure of rules will vary depending on the Guard, the resources the guard protects, the structure of Subject Descriptors, naming of resources, and other factors.

Collaborations

- If the Client is operating on behalf of a specific user, and that user's security attributes have not previously been retrieved and cached, the Client may invoke the Authenticator to establish and verify that user's identity.
- If used, the Authenticator causes the user's security attributes to be included in the effective Security Context.
- The Client submits an access request to the Guard.
- The Guard determines the request, target, and context attributes.
- The Guard requests a policy decision (passing in the Client, request, target, and context attributes) from the Policy.
- The Policy checks which of the set of Rules matches the security attributes, requested operation, and the targeted Resource.
- If the Rules indicate that the request should be denied, the Policy returns a notification to the Guard, which then passes a failure notification to the Client.
- If the Policy result indicated that the request should be granted, the Guard passes the request for fulfillment to the Resource and relays the response back to the Client.



Consequences

Use of the Policy pattern:

- Loosens coupling between policy enforcement and resource implementation.

  This permits resource managers to be built with no awareness of authentication, attribute collection, policy evaluation, and policy enforcement.

- Ensures that security policy is checked before client requests for resources are fulfilled.

  Sequencing security policy checks and resource request fulfillment can be used to ensure that policy checks are always performed in the correct order, and that they are always performed before resource requests are fulfilled.

- Provides a single point of control and management for policy-related activities.

  Localizing policy-related operations improves assurability of the system by limiting the amount of system code which needs to be examined during assurance activities.

  Localizing policy-related operations may also create a single point of failure or attack; designers should take care to address these problems by hardening the Policy component and by addressing availability of the Policy (perhaps by use of the Redundant System pattern).

- Requires matching of the Rules parameter signature with resource namespace and operation signatures.

  The policy expressed by the Rules needs to ‘‘speak the same language’’ as the Guard uses when checking requests for access to resources.

  Adding new resources or new operations to a system without changing the Rules policy language and enforcement capability can weaken the protection provided by the Guard.

  For example, adding relational queries to a system whose Policy recognizes and protects only the files in which the relational tables are stored will leave the system vulnerable to unauthorized disclosures of information through inference.

- Imposes performance overhead.

  Separating policy enforcement from resource request fulfillment will usually introduce additional procedure calls or network overhead.

Implementation

- Time-of-check *versus* time-of-use

  Many policies are time-sensitive. Security authorization policies, for example, are sensitive to the status of a user’s account.

  In a system which uses the Policy pattern to separate policy enforcement from resource access request fulfillment, there will be a delay between evaluation of policy and fulfillment of requests. If this delay is long, there is a possibility that the user’s authorization status will have changed after the policy is evaluated but before the request is fulfilled (for example, the user’s account may have been suspended or revoked).

  Designers using the Policy pattern should take care to minimize the interval between the time the Policy makes a decision and the time the request is fulfilled by the Resource.

- Policy and Rule interface design

  Designing Policies which can be extended to support new types of resources and new operations on existing resources (so that it is not necessary to replace the system's Policy whenever a new type or version of a Resource is added to the system) is difficult. Very broadly extensible Policies are also very hard to assure, because of the difficulty of analyzing the policy which is actually enforced.

  In general, Policies consisting of only monotonic Rules (such as Rules which only express the granting of access, with denial of access being represented by an absence of any matching Rules) are easier to manage and assure.

  As it is a characteristic of a Protected System that a single Policy makes all access decisions, use of the Singleton [GoF] pattern may be appropriate to ensure that only a single Policy object will be instantiated.

  Rules are commonly expressed as triplets of (subject, operation, resource).

Known Uses

Firewall Routing Filter, Content Scanner, Reference Monitor, Credit Authorization

CORBASecurity provides three instances of Policy:

1. Client Secure Invocation Policies describe the client's minimum requirement for Quality of Protection (QoP) when initiating a Secure Communication.
2. Server Secure Invocation Policies define the range of Quality of Protection options which are supported when accepting a Secure Communication and also the minimum requirement for Quality of Protection when accepting a Secure Communication.
3. Object References contain a copy of the object's Server Secure Invocation Policy information.

Related Patterns

- Protected System [TG_SDP]
- Redundant System [TG_SDP]

  Introducing a centralized Policy implementation may, if done carelessly, create a single point of failure in a system which otherwise would not have one. Use of the Redundant System pattern is recommended to avoid this happening.
- Authenticator [Brown-Fernandez]
- Security models, authorization, multi-level security [Fernandez-Pan]
- Security policy [Mahmoud]