Gateway API: Authorization Policy API Design Options for TargetRef

This document presents design options for implementing authorization policies in Gateway API, specifically targeting East/West (in-cluster) traffic for Mesh implementations. The goal is to create a vendor-neutral API that works across different mesh architectures while addressing the unique constraints of sidecar-based and ambient mesh implementations.

Mesh Implementation Types

Sidecar-Based Meshes

- **Architecture**: Sidecar proxy deployed alongside every pod
- **Capabilities**: Can enforce all request attributes (L4 and L7)
- **Targeting**: Uses label selectors to target sidecars for policy distribution
- **Enforcement Point**: Single enforcement point (the sidecar)

Ambient Meshes

- **Architecture**: Two-tier proxy system
    - Node-L4-proxy: Handles identity-based policies and port enforcement
    - L7 enforcement point (waypoint proxy): Handles advanced L7 features
- **Targeting**:
    - Label selectors for node proxy distribution
    - Service targetRef for L7 enforcement at waypoints
    - **Constraint**: Selectors and targetRef cannot be used together

Key Challenges

1. **Architectural Differences**: Ambient meshes separate L4 and L7 enforcement, while sidecars handle both
2. **Targeting Mechanisms**:
    - Ambient struggles with label selectors for L7 enforcement on waypoints
    - Sidecars have difficulty supporting Service targeting for L7 enforcement
3. **API Consistency**: Need a unified approach that works across all implementations

Design Options

Option 1: Separate L4 and L7 APIs

Overview

Create two distinct APIs:

- **L4 Authorization Policy**: Handles identity-based and port-based authorization
- **L7 Authorization Policy**: Superset of L4 with additional L7 features (headers, paths, methods, etc.)

Implementation Details

L4 Authorization Policy

```
None
apiVersion: gateway.networking.k8s.io/v1alpha1
kind: L4AuthorizationPolicy
metadata:
  name: l4-authz-policy
spec:
  targetRef:
    kind: Pod
    selector: {}
  rules:
  - authorizationSource:
      serviceAccount: ["default/productpage"]
    tcpAttributes:
      ports: [9080]
```

L7 Authorization Policy

```
None
apiVersion: gateway.networking.k8s.io/v1alpha1
kind: L7AuthorizationPolicy
metadata:
  name: l7-authz-policy
spec:
```

```
targetRef:
  # Label selector or Service reference
rules:
- authorizationSource:
    serviceAccount: ["default/productpage"]
  tcpAttributes:
    ports: [9080]
  httpAttributes:
    paths: ["/api"]
    methods: ["GET", "POST"]
```

Advantages

- **Broad Compatibility**: L4 API implementable by almost every mesh
- **Baseline Portability**: Provides consistent L4 authorization across implementations
- **Ambient Optimization**: Convenient for ambient meshes when waypoints aren't involved
- **Clear Separation**: Distinct APIs for distinct enforcement points
- **Implementation Flexibility**: Each mesh can optimize for its architecture

Disadvantages

- **Migration Complexity**: Users upgrading from L4 to L7 must:
  - Create new L7 policy
  - Potentially change targetRef
  - Remove old L4 policy
  - Risk of policy enforcement gaps during transition
- **User Experience**: 70%+ of mesh users need L7 features, making separate APIs potentially cumbersome
- **Policy Management**: Harder to understand which policies are actually enforced
- **Operational Overhead**: More APIs to manage and understand

Option 2: Single API with Validation

Overview

Create one unified authorization policy API with implementation-specific validation using Validating Admission Policies (VAP).

Implementation Details

Unified Authorization Policy

```
None
apiVersion: gateway.networking.k8s.io/v1alpha1
kind: AuthorizationPolicy
metadata:
  name: unified-authz-policy
spec:
  targetRef: # each mesh implementation bundles with VAP
     # label selector for sidecar and l4-ambient
     # Service for L7 ambient
  rules:
    - authorizationSource:
        serviceAccount: ["default/productpage"]
      tcpAttributes:
        ports: [9080]
      httpAttributes:
        paths: ["/api"]
        methods: ["GET", "POST"]
```

Validation Logic

- **Sidecar Implementation VAP**: Validates that targetRef is not a Service
- **Ambient Implementation VAP**: Validates that if targetRef uses label selectors, only L4 attributes are specified

Advantages

- **Unified Experience**: Single API for all authorization needs
- **Smooth Migration**: Users can add L7 features to existing policies without creating new ones
- **Familiar Pattern**: Matches current mesh user expectations
- **Simplified Management**: One API to understand and manage
- **Flexible Enforcement**: Implementations can enforce at appropriate points

Disadvantages

- **Validation Complexity**: Requires sophisticated VAP rules
- **Mixed Cluster Challenges**: Potential confusion with both sidecar and ambient workloads
- **Implementation Burden**: Each mesh must implement validation logic

- **Error Handling**: More complex error scenarios and status reporting

Option 3: Different Scoping

What if this will be about **where** the policy applies rather than **what** it contains?

```
None
# Network-scoped policy (L4 enforcement points)
kind: AuthorizationPolicy
metadata:
  name: network-authz
spec:
  enforcementLevel: "network"  # Enforced at L4 proxies
  targetRef: # Label selectors work here
  rules:
    - authorizationSource:
        serviceAccount: ["default/productpage"]
      tcpAttributes:
        ports: [9080]

# Application-scoped policy (L7-only enforcement points and
sidecars)
kind: AuthorizationPolicy
metadata:
  name: app-authz
spec:
  enforcementLevel: "application"  # Enforced at L7 proxies
  targetRef:  # each mesh implementation bundles with VAP
     # label selector for sidecar
     # Service for L7 ambient

  rules:
    - authorizationSource:
        serviceAccount: ["default/productpage"]
      tcpAttributes:
        ports: [9080]
```

```
      httpAttributes:
        paths: ["/api/"]
        methods: ["GET", "POST"]
```

For Ambient Mesh

```
None
# This would be enforced at node-l4-proxy
kind: AuthorizationPolicy
spec:
  scope: "network"
  targetRef:
   Kind: Pod
      Selector: {}
  rules: # Only L4 features allowed

# This would be enforced at waypoint
kind: AuthorizationPolicy
spec:
  enforcementLevel: "application"
  targetRef:
   Kind: Service # Works with ambient waypoints
   Name: foo
  rules: # L4 + L7 features allowed
```

For Sidecar Mesh

```
None
# Both enforced at sidecar, but different feature sets
kind: AuthorizationPolicy
spec:
```

```
  enforcementLevel: "network"    # Sidecar enforces, but only L4
features
  enforcementLevel: "application" # Sidecar enforces with full
L4+L7 features
```

Conclusion

Single API with Validation and enforcementLevel

Create one unified authorization policy API with implementation-specific validation using Validating Admission Policies (VAP). Additionally introducing a new `enforcementLevel` enum to simplify validation and enable the user to clarify intent

```
None
# Network-scoped policy (L4 enforcement points)
kind: AuthorizationPolicy
metadata:
  name: network-authz
spec:
  enforcementLevel: "network"  # Enforced at L4 proxies
  targetRef:
    kind: Pod
    selector: {}
  rules:
    - authorizationSource:
        serviceAccount: ["default/productpage"]
      tcpAttributes:
        ports: [9080]

# Application-scoped policy (L7-only enforcement points and
sidecars)
kind: AuthorizationPolicy
```

```
metadata:
  name: app-authz
spec:
  enforcementLevel: "application"  # Enforced at L7 proxies
  targetRef: # Service for Ambient implementations, label-selector
for sidecar implementations.

  rules:
    - authorizationSource:
        serviceAccount: ["default/productpage"]
      tcpAttributes:
        ports: [9080]
      httpAttributes:
        paths: ["/api"]
        methods: ["GET", "POST"]
```

Mesh Implementation Behavior

Sidecar Meshes

- **Network Level**: Sidecar enforces policy but only uses L4 attributes
- **Application Level**: Sidecar enforces policy with full L4+L7 capabilities

Ambient Meshes

- **Network Level**: Node-L4-proxy enforces policy using labelSelector targeting
- **Application Level**: Waypoint proxy enforces policy using service targetRef targeting

VAP Validation:

  - **Sidecar mesh**: Ensures application-level policies don't use targetRef: Service (since sidecars use labelSelector)
  - **Ambient**: Ensures application-level policies don't use label-selectors.
  - **Both sidecar and ambient:** ensures network level policies **do** use label selectors

Advantages

- **Clear Intent**: Users explicitly state where they want enforcement
- **Flexible Targeting**: Different targetRef types for different enforcement points
- **Migration Path**: Users can start with network-level and upgrade to application-level easily
- **Implementation Alignment**: supporting different meshes architectures
- **Single API**: No duplication of schemas or concepts
- **Validation Clarity**: Clear rules about what's allowed at each level

Disadvantages

- **Complexity**: Users must understand enforcement levels
- **VAP Dependency**: Requires validation rules