Security Architecture

Comprehensive Security Framework for Service Function Architecture Platform

---

Table of Contents

Executive Summary

This document defines the security architecture for the Service Function Architecture platform. The security model is built on orthogonal components that provide defense-in-depth protection while maintaining the platform's simplicity and performance characteristics.

Security Principles:

- **Zero Trust Architecture**: No implicit trust based on network location or system identity
- **Principle of Least Privilege**: Users and services receive minimum necessary permissions
- **Defense in Depth**: Multiple independent security layers protect against different attack vectors
- **Security by Design**: Security controls integrated into platform architecture, not bolted on
- **Transparency and Auditability**: All security decisions logged and traceable

Core Components:

- **Identity and Authentication**: OIDC integration with enterprise identity providers
- **Authorization Framework**: Fine-grained RBAC with resource-level permissions
- **Multi-Tenant Isolation**: Secure tenant boundaries with data and process isolation
- **Message Security**: Topic-level access control and message integrity protection
- **Network Security**: Encrypted communication and network segmentation
- **Audit and Compliance**: Comprehensive logging for regulatory requirements

---

Security Architecture Overview

Layered Security Model

The platform implements security through orthogonal layers that operate independently:

Identity Layer: Authenticates users and services using OIDC tokens and mutual TLS certificates.

Authorization Layer: Enforces fine-grained permissions at resource and operation levels through RBAC policies.

Communication Layer: Secures all message traffic through topic-level access controls and optional message encryption.

Data Layer: Protects stored data through encryption at rest, row-level security, and tenant isolation.

Network Layer: Enforces network segmentation, TLS encryption, and firewall policies.

Audit Layer: Logs all security-relevant events for compliance and threat detection.

Security Boundaries

Trust Boundaries:

- **External Users**: Web and mobile applications connecting via authenticated MQTT
- **Internal Services**: Service functions running in isolated containers with service credentials
- **Administrative Access**: Operations and management interfaces with elevated privileges
- **Infrastructure**: Platform components with system-level access

Isolation Boundaries:

- **Tenant Isolation**: Complete separation of multi-tenant data and processes
- **Service Isolation**: Container-level isolation between different service functions
- **User Isolation**: Session-based isolation for user-specific data and communications
- **Network Isolation**: Network segmentation between platform tiers

---

Identity and Authentication

OIDC Integration Architecture

Identity Provider Integration: The platform integrates with enterprise identity providers through standard OIDC protocols, supporting:

- Azure Active Directory
- Google Workspace
- Okta
- Auth0
- Custom OIDC-compliant providers

Token Flow:

1. **User Authentication**: Users authenticate with their identity provider
2. **Token Issuance**: Identity provider issues JWT access tokens with user claims
3. **Token Presentation**: Clients include bearer tokens in MQTT message headers
4. **Token Validation**: Platform services validate tokens using provider public keys
5. **User Context Creation**: Validated tokens create UserContext objects for authorization

UserContext Structure

Core Identity Information:

```csharp
public class UserContext
{
    public string UserId { get; set; }       // Unique user identifier
    public string Email { get; set; }        // User email address
    public string SessionId { get; set; }    // Current session identifier
    public List<string> Roles { get; set; }  // Assigned roles
    public List<string> Groups { get; set; } // Group memberships
    public string Token { get; set; }        // Original OIDC token
    public string TenantId { get; set; }     // Multi-tenant identifier
    public Dictionary<string, object> Claims { get; set; } // Additional OIDC claims

    public bool HasRole(string role) => Roles.Contains(role);
    public bool HasGroup(string group) => Groups.Contains(group);
}
```

Token Validation Process: Service functions automatically validate OIDC tokens through the platform's authentication interface:

1. **Extract Token**: Parse bearer token from message headers
2. **Signature Verification**: Validate JWT signature using provider's public key
3. **Claims Validation**: Verify token expiration, audience, and issuer claims
4. **User Mapping**: Map OIDC claims to platform user context
5. **Session Tracking**: Associate validated context with user session

Service-to-Service Authentication

Mutual TLS Authentication: Internal service communication uses mutual TLS certificates for authentication:

- Each service function receives unique TLS client certificates
- Certificate-based authentication for high-security inter-service calls
- Automatic certificate rotation through Kubernetes cert-manager

API Key Authentication: For performance-critical internal communication:

- Services receive unique API keys for identification
- Keys included in `x-api-key` message headers
- Regular key rotation with zero-downtime updates
- Key-based rate limiting and abuse detection

Authorization Framework

Role-Based Access Control (RBAC)

Permission Model: The platform uses a hierarchical RBAC model with permissions granted at multiple levels:

System Level: Platform-wide administrative permissions

- `platform:admin` - Full platform administration
- `platform:operator` - Infrastructure management
- `platform:viewer` - Read-only system access

Service Level: Service function access permissions

- `service:UserManagement:invoke` - Can call UserManagement functions
- `service:*:invoke` - Can call any service function
- `service:PaymentService:admin` - Administrative access to PaymentService

Data Level: Database and collection access permissions

- `data:users:read` - Read access to users collection
- `data:orders:write` - Write access to orders collection
- `data:*:admin` - Administrative access to all data

Topic Level: MQTT topic access permissions

- `topic:ui/change/{userId}:publish` - Can publish UI changes for specific user
- `topic:session/{sessionId}:subscribe` - Can subscribe to specific session topics
- `topic:system/*:subscribe` - Can subscribe to system topics

Authorization Interface Integration

IAuthorizer Implementation: The authorization framework integrates with platform interfaces to provide seamless permission checking:

```csharp
// Role assignment and management
await authorizer.AssignRoleToUser("user123", "service-developer", "platform");
await authorizer.RemoveRoleFromUser("user123", "admin", "platform");

// Permission checking
bool canRead = await authorizer.HasPermission("user123", "data:users:read", "platform");
bool canManage = await authorizer.HasRole("user123", "service-admin", "platform");

// Fine-grained resource permissions
bool canEditOrder = await authorizer.HasPermission("user123", "data:orders:write", "tenant-456", "order-789"
```

Dynamic Permission Evaluation: Permissions are evaluated in real-time based on:

- User's assigned roles and groups
- Resource ownership and sharing rules
- Time-based access restrictions
- Context-dependent policies (location, device, etc.)

Multi-Tenant Authorization

Tenant Isolation Model: Multi-tenant deployments enforce strict authorization boundaries:

Tenant-Scoped Permissions:

- All permissions include tenant context
- Users can only access resources within authorized tenants
- Cross-tenant access requires explicit inter-tenant permissions

Resource Namespacing:

- Database collections prefixed with tenant identifier
- MQTT topics scoped to tenant namespace
- Service functions operate within tenant context

Administrative Boundaries:

- Tenant administrators cannot access other tenants
- Platform administrators can access all tenants
- Service administrators operate within specific tenant scope

---

Message Security

Topic-Level Access Control

Topic Permission Patterns: Message security operates through topic-level permissions that map directly to the platform's topic hierarchy:

Service Function Access:

- `topic:service/UserManagement:publish` - Can invoke UserManagement service
- `topic:service/*:publish` - Can invoke any service function
- `topic:service/OrderManagement:admin` - Administrative access to OrderManagement

Session Access:

- `topic:session/{sessionId}/*:subscribe` - Can receive messages for specific session
- `topic:session/*:publish` - Can send responses to any session (service functions only)

UI Update Access:

- `topic:ui/change/{userId}:subscribe` - Can receive UI updates for specific user
- `topic:ui/change/group/{groupId}:subscribe` - Can receive group UI updates
- `topic:ui/change/*:publish` - Can trigger UI updates (services only)

System Access:

- `topic:system/*/errors:subscribe` - Can receive system error notifications
- `topic:system/*:publish` - Can publish system messages (infrastructure only)

Message Integrity and Encryption

Message Signing: Critical messages include cryptographic signatures for integrity verification:

- HMAC signatures using shared secrets for performance
- Digital signatures using RSA/ECDSA for high-security scenarios
- Automatic signature verification in message processing pipeline

Optional Message Encryption: For sensitive data transmission:

- AES-256 encryption for message payloads
- Key management through platform secrets system
- End-to-end encryption for user-to-user communication

Message Headers Security: Authentication and authorization information travels in message headers:

- Bearer tokens for user authentication
- API keys for service authentication
- Message signatures for integrity
- Encryption metadata for secure messages

Data Security

Encryption at Rest

Database Encryption:

- PostgreSQL: Transparent Data Encryption (TDE) for tablespaces
- MongoDB: WiredTiger encryption for collections and indexes
- Redis: Memory encryption for sensitive cached data
- File Storage: AES-256 encryption for stored files

Key Management:

- Hardware Security Modules (HSM) for key generation and storage
- Automatic key rotation on configurable schedules
- Separate encryption keys per tenant for multi-tenant deployments
- Key escrow and recovery procedures for compliance

Row-Level Security

PostgreSQL RLS Implementation: Row-level security policies automatically filter data based on user context:

```sql
-- User can only see their own records
CREATE POLICY user_isolation ON users
  USING (user_id = current_setting('app.current_user_id'));

-- Tenant isolation for multi-tenant tables
CREATE POLICY tenant_isolation ON orders
  USING (tenant_id = current_setting('app.current_tenant_id'));

-- Role-based access to sensitive data
CREATE POLICY admin_access ON audit_logs
  USING (current_setting('app.user_role') = 'admin');
```

MongoDB Access Control:

- Field-level access control through projection filters
- Document-level filtering based on user context
- Tenant-scoped database access with automatic filtering

Data Classification and Handling

Data Classification Levels:

- **Public**: No access restrictions (product catalogs, public documentation)
- **Internal**: Organization members only (internal processes, system data)
- **Confidential**: Restricted access (user personal data, business records)
- **Restricted**: Highly sensitive (payment data, authentication credentials)

Handling Requirements:

- Restricted data requires encryption in transit and at rest
- Confidential data requires audit logging for all access
- Internal data requires authentication and authorization
- Public data has no special requirements

Network Security

Network Segmentation

Three-Tier Architecture: The platform implements network segmentation through three distinct tiers:

DMZ Tier (10.0.1.0/24):

- External load balancers and reverse proxies
- Web application firewall (WAF)
- TLS termination points
- External monitoring and management interfaces

Application Tier (10.0.2.0/24):

- Service function containers
- MQTT message brokers
- Application load balancers
- Internal API gateways

Data Tier (10.0.3.0/24):

- Database clusters (PostgreSQL, MongoDB, Redis)
- Object storage systems
- Backup and archival storage
- Internal monitoring databases

Encryption in Transit

TLS Configuration: All network communication uses TLS 1.3 with strong cipher suites:

- MQTT connections: TLS 1.3 with mutual authentication
- Database connections: TLS with certificate pinning
- Service-to-service: Mutual TLS with automatic certificate rotation
- External APIs: TLS 1.3 with perfect forward secrecy

Certificate Management:

- Automatic certificate provisioning through cert-manager
- Regular certificate rotation (90-day lifecycle)
- Certificate transparency logging
- OCSP stapling for revocation checking

Firewall and Access Control

Network Policies: Kubernetes network policies enforce micro-segmentation:

```yaml
# Service functions can only access databases and message brokers
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: service-function-policy
spec:
 podSelector:
  matchLabels:
   tier: service-function
 policyTypes:
 - Ingress
 - Egress
 egress:
 - to:
  - podSelector:
     matchLabels:
      tier: database
  ports:
  - protocol: TCP
   port: 5432
 - to:
  - podSelector:
     matchLabels:
      tier: messaging
  ports:
  - protocol: TCP
   port: 1883
```

External Firewall Rules:

- Only necessary ports exposed externally (443 for HTTPS, 8883 for MQTT over TLS)
- Geographic IP filtering for administrative access
- DDoS protection and rate limiting
- Intrusion detection and prevention systems

Audit and Compliance

Comprehensive Audit Logging

Audit Event Categories: All security-relevant events are logged with appropriate detail:

Authentication Events:

- User login attempts (successful and failed)
- Token validation and renewal
- Service authentication events
- Multi-factor authentication usage

Authorization Events:

- Permission grants and denials
- Role assignments and modifications
- Policy changes and updates
- Privilege escalation attempts

Data Access Events:

- Database queries and modifications
- File access and transfers
- API calls and responses
- Data exports and imports

System Events:

- Configuration changes
- Security policy updates
- Certificate operations
- Administrative actions

Audit Log Format

Structured Logging: All audit events follow a consistent JSON structure for parsing and analysis:

```json
{
  "timestamp": "2024-12-15T10:30:00Z",
  "eventType": "authentication",
  "action": "user_login",
  "result": "success",
  "userId": "user123",
  "sessionId": "sess_abc123",
  "sourceIp": "192.168.1.100",
  "userAgent": "Platform-Client/1.2.0",
  "tenantId": "tenant-456",
  "resource": "platform",
  "details": {
    "authMethod": "oidc",
    "provider": "azure-ad",
    "mfaUsed": true
  },
  "correlationId": "req-789"
}
```

Compliance Framework

Regulatory Standards: The platform supports compliance with major regulatory frameworks:

GDPR (General Data Protection Regulation):

- Data subject rights implementation (access, rectification, erasure)
- Consent management and tracking
- Data processing transparency and logging
- Data protection impact assessments

SOX (Sarbanes-Oxley Act):

- Financial data access controls
- Change management procedures
- Audit trail integrity
- Segregation of duties

HIPAA (Health Insurance Portability and Accountability Act):

- Protected health information safeguards
- Access controls and audit logging
- Data encryption requirements
- Breach notification procedures

SOC 2 (Service Organization Control 2):

- Security principle compliance
- Availability and processing integrity
- Confidentiality controls
- Privacy protection measures

Data Retention and Purging

Retention Policies:

- Security logs: 7 years for compliance requirements
- Audit trails: 3 years for operational analysis
- Session data: 90 days for security investigation
- Performance logs: 30 days for optimization

Automated Purging:

- Scheduled deletion of expired data
- Secure data destruction procedures
- Retention policy enforcement
- Legal hold capabilities for litigation

Threat Protection

Threat Detection and Response

Behavioral Analytics: The platform monitors user and system behavior to detect anomalies:

- Unusual login patterns and locations
- Abnormal data access patterns
- Suspicious API usage patterns
- Privilege escalation attempts

Automated Response: Detected threats trigger automatic protective actions:

- Account lockout for brute force attempts
- Session termination for suspicious activity
- Rate limiting for unusual traffic patterns
- Alert generation for security team investigation

Vulnerability Management

Security Scanning:

- Container image vulnerability scanning
- Dependency vulnerability assessment
- Static code analysis for security issues
- Dynamic application security testing

Patch Management:

- Regular security updates for all components
- Automated patching for critical vulnerabilities
- Staged rollout for compatibility testing
- Emergency patching procedures for zero-day threats

Incident Response Integration

Security Incident Workflow:

1. **Detection**: Automated monitoring and alerting systems
2. **Analysis**: Security team investigates alerts and logs
3. **Containment**: Isolate affected systems and users
4. **Eradication**: Remove threats and close vulnerabilities
5. **Recovery**: Restore normal operations and monitor
6. **Lessons Learned**: Update procedures and controls

Integration with Operations: Security incidents integrate with the platform's operational incident response:

- Security alerts routed through same incident management system
- Coordinated response between security and operations teams
- Unified communication and escalation procedures
- Post-incident review process includes security analysis

Security Configuration Management

Security Hardening

Container Security:

- Minimal base images with no unnecessary packages
- Non-root container execution
- Read-only root filesystems
- Resource limits and security contexts
- Regular security scanning and updates

Kubernetes Security:

- Pod Security Standards enforcement
- Network policies for micro-segmentation
- RBAC with principle of least privilege
- Secrets management through external systems
- Admission controllers for policy enforcement

Security Monitoring

Real-Time Monitoring:

- Failed authentication attempt tracking
- Unusual access pattern detection
- Certificate expiration monitoring
- Vulnerability scan result analysis
- Compliance status tracking

Security Dashboards:

- Authentication success/failure rates
- Authorization decision metrics
- Security event trending
- Compliance posture status
- Threat detection alerts

Configuration Validation

Security Policy as Code: Security configurations are managed through Infrastructure as Code:

- Version-controlled security policies
- Automated policy deployment
- Configuration drift detection
- Policy compliance validation
- Change approval workflows

Regular Security Assessments:

- Quarterly penetration testing
- Annual security architecture reviews
- Continuous compliance monitoring
- Third-party security audits
- Red team exercises

Multi-Tenant Security

Tenant Isolation Architecture

Complete Data Separation: Each tenant operates in a completely isolated environment:

- Separate database schemas or instances per tenant
- Tenant-scoped MQTT topic namespaces
- Isolated service function execution contexts
- Separate encryption keys per tenant

Cross-Tenant Security:

- No shared data structures between tenants
- Network isolation between tenant workloads
- Separate authentication and authorization contexts
- Independent audit trails per tenant

Tenant Administration

Tenant-Level Administration: Each tenant has independent administrative capabilities:

- Tenant administrators cannot access other tenants
- Self-service user and role management
- Tenant-specific security policies
- Independent compliance and audit reporting

Platform Administration: Platform administrators have controlled cross-tenant access:

- Limited to operational and security functions
- All cross-tenant access logged and audited
- Separate authentication for platform admin functions
- Emergency access procedures with approval workflows

Security Operations

Security Team Structure

Security Roles and Responsibilities:

- **Security Architect**: Overall security design and strategy
- **Security Engineer**: Implementation and tooling
- **Security Analyst**: Monitoring and incident response
- **Compliance Officer**: Regulatory compliance and auditing

Security Operations Center (SOC):

- 24/7 monitoring and incident response
- Threat intelligence analysis
- Security event correlation and analysis
- Incident escalation and coordination

Security Metrics and KPIs

Security Performance Indicators:

- Mean time to detect (MTTD) security incidents
- Mean time to respond (MTTR) to security events
- Authentication failure rates and patterns
- Vulnerability remediation timelines
- Compliance audit results and findings

Security Dashboard Metrics:

- Active security alerts and incidents
- Authentication and authorization metrics
- Security policy compliance status
- Vulnerability scan results and trends
- Security training completion rates

Conclusion

The Security Architecture provides comprehensive protection for the Service Function Architecture platform through layered, orthogonal security controls. The design balances strong security with operational simplicity, ensuring that security enhances rather than hinders platform capabilities.

Key Security Strengths:

- **Zero Trust Model**: No implicit trust based on network location
- **Defense in Depth**: Multiple independent security layers
- **Fine-Grained Access Control**: Resource-level permissions with real-time evaluation
- **Comprehensive Audit Trail**: Complete logging for compliance and investigation
- **Multi-Tenant Isolation**: Secure boundaries between tenants
- **Automated Threat Response**: Real-time detection and response capabilities

Operational Benefits:

- **Integrated Security**: Security controls built into platform architecture
- **Simplified Management**: Unified security model across all platform components
- **Compliance Ready**: Built-in support for major regulatory frameworks
- **Scalable Security**: Security controls scale with platform growth
- **Developer Friendly**: Security integration transparent to service function developers

This security architecture ensures that the platform meets enterprise security requirements while maintaining the simplicity and performance characteristics that make the Service Function Architecture effective for rapid application development and deployment.