PII Detection & Redaction Deployment Strategy

Executive Summary

Based on the architecture diagram and the identified security breach scenario, I propose deploying the PII detection solution as a **multi-layer approach** with the primary component being an **API Gateway Plugin** integrated with **sidecar containers** for microservices.

Proposed Architecture

Primary Deployment: API Gateway Plugin

Location: At the API Routes Proxy layer (shown in the architecture diagram)

Rationale:

- **Single Point of Control**: All external API calls pass through this layer
- **Low Latency**: Processes data in-flight without additional network hops
- **Centralized Management**: One deployment point for the entire platform
- **Cost Effective**: No need for separate infrastructure

Secondary Deployment: Sidecar Containers

Location: Alongside microservices in the backend (Express + MCP)

Rationale:

- **Deep Integration**: Catches PII that might be generated internally
- **Service-Specific Rules**: Can customize PII detection per service
- **Backup Layer**: Ensures coverage if API gateway layer fails

Technical Implementation

1. API Gateway Plugin

An **External Request** first goes into the **PII Plugin (Fast Path)**, where sensitive data can be detected and redacted. From there, the sanitized request is passed along to the **Backend Services** for normal processing.

Technology Stack:

- **Language**: Go (for performance) or Python (for rapid development)
- **Framework**: Kong/Nginx plugin architecture
- **Processing**: Streaming JSON parser for real-time redaction
- **Caching**: Redis for pattern matching cache

Key Features:

- Sub-millisecond processing for most requests
- Configurable sensitivity levels per endpoint
- Async logging of PII detection events
- Circuit breaker for high availability

2. Sidecar Container Implementation

The Main Service Container communicates directly with the PII Sidecar Container. Both run alongside each other, and data flows back and forth between them.

Features:

- Lightweight HTTP proxy (using Envoy)
- Shared volume for configuration updates
- Health check integration
- Auto-scaling based on traffic

Deployment Locations & Justification

Layer 1: Network Ingress (Primary)

- **Where**: API Routes Proxy
- **Why**: Catches external threats, handles 80% of PII exposure
- **Latency Impact**: <5ms additional processing time
- **Coverage**: All external API traffic

Layer 2: Application Layer (Secondary)

- **Where**: Microservice sidecars
- **Why**: Internal PII generation, service-to-service communication
- **Latency Impact**: <2ms (local container communication)
- **Coverage**: Internal data flows

Layer 3: Data Layer (Monitoring)

- **Where**: Database proxy layer
- **Why**: Final safety net, audit trail
- **Latency Impact**: Async processing (no real-time impact)

- **Coverage**: Data persistence layer

Scalability & Performance

Horizontal Scaling

- API Gateway: Load balancer distributes traffic
- Sidecars: Auto-scale with parent services
- Monitoring: Event-driven scaling

Performance Optimizations

1. **Regex Compilation**: Pre-compiled patterns cached in memory
2. **Streaming Processing**: JSON streaming parser to avoid large memory allocation
3. **Async Operations**: Non-blocking PII logging and alerting
4. **Smart Sampling**: Process 100% of suspicious traffic, sample normal traffic

Expected Performance Metrics

- **Throughput**: 10,000+ requests/second per gateway instance
- **Latency**: 95th percentile <10ms additional overhead
- **Memory**: <512MB per instance
- **CPU**: <20% overhead during peak traffic

Cost Analysis

Infrastructure Costs (Monthly)

- API Gateway Plugin: $200 (leverages existing infrastructure)
- Sidecar Containers: $800 (additional container resources)
- Monitoring & Logging: $300 (storage and processing)
- **Total**: ~$1,300/month

Cost Savings

- **Prevented Fraud**: $50,000+ per incident prevented
- **Compliance**: Avoid regulatory fines ($100k+ potential)
- **ROI**: Break-even within first prevented incident

Integration Strategy

Phase 1: Proof of Concept (2 weeks)

1. Deploy plugin on staging API gateway

2. Test with sample traffic
3. Measure performance impact
4. Fine-tune detection rules

Phase 2: Gradual Rollout (4 weeks)

1. Deploy on 10% of production traffic
2. Monitor false positives/negatives
3. Adjust sensitivity settings
4. Scale to 100% of traffic

Phase 3: Full Coverage (2 weeks)

1. Deploy sidecar containers
2. Enable database monitoring
3. Complete audit trail implementation
4. Staff training on monitoring tools

Monitoring & Alerting

Key Metrics

- PII detection rate (per endpoint)
- False positive ratio
- Processing latency
- System resource utilization

Alert Thresholds

- **Critical**: >1000 PII detections/hour from single source
- **Warning**: >50 false positives/hour
- **Info**: Performance degradation >10ms average

Risk Mitigation

High Availability

- Multi-zone deployment
- Circuit breaker patterns
- Graceful degradation (fail-open with logging)

Security

- Encrypted PII logs
- Limited retention (30 days)

- Access control for detection rules
- Regular security audits

Alternative Considered

Browser Extension

Pros: Catches client-side leaks **Cons**: Limited coverage, harder to manage, performance issues **Decision**: Not selected due to scalability concerns

Database Triggers

Pros: Catches all data persistence **Cons**: High latency, database performance impact **Decision**: Used only for monitoring/auditing

Conclusion

The proposed multi-layer approach with primary deployment at the API Gateway provides optimal balance of:

- **Performance**: Minimal latency impact
- **Coverage**: Comprehensive PII detection
- **Cost**: Leverages existing infrastructure
- **Maintenance**: Centralized management

This solution directly addresses the breach scenario by catching PII leaks at the network ingress layer before they can reach external logs or unmonitored endpoints.