# OBINexus Zero Trust Topology UML Architecture

## Component Node-Network Defense Architecture

```
mermaid
```

```
graph TB
    subgraph "Core Defense Layer"
        CoreNode[Core Protocol Node]
        CoreValidator[Cryptographic Validator]
        CoreManifest[XML Manifest Controller]
    end

    subgraph "CLI Defense Layer"
        CLIGateway[CLI Gateway Node]
        CLIValidator[CLI Command Validator]
        CLIManifest[CLI Manifest Processor]
    end

    subgraph "Build Defense Layer"
        BuildOrchestrator[Build Orchestrator Node]
        BuildValidator[Build Integrity Validator]
        BuildManifest[Build Manifest Generator]
    end

    subgraph "Governance Defense Layer"
        GovValidator[Governance Validator Node]
        GovPolicy[Policy Enforcement Engine]
        GovAudit[Audit Trail Controller]
    end

    subgraph "Source Repository Topology"
        SrcRoot[src/ Root Directory]
        SrcInput[*.in Source Files]
        SrcCrypt[*.crypt Security Files]
        SrcPriv[*.privation Access Control]
        SrcPub[*.pub Public Keys]
    end

    subgraph "Manifest Management Topology"
        ManifestRoot[root/src Manifest Storage]
        ManifestBuild[build/.xml Generated Manifests]
        ManifestPublic[Public Manifest Registry]
        ManifestPrivate[Private Manifest Vault]
    end

    subgraph "Zero Trust Validation Network"
        ZTValidator[Zero Trust Validator]
        PhantomEncoder[Phantom Encoder Node]
        CryptoProof[Cryptographic Proof Engine]
        StateManager[Gated State Manager]
    end
```

```
%% Core Defense Connections
CoreNode -->|Encrypted Protocol| CoreValidator
CoreValidator -->|Verified State| CoreManifest
CoreManifest -->|Manifest Output| ManifestRoot

%% CLI Defense Connections
CLIGateway -->|Command Validation| CLIValidator
CLIValidator -->|Sanitized Commands| CLIManifest
CLIManifest -->|CLI Manifest| ManifestBuild

%% Build Defense Connections
BuildOrchestrator -->|Build Request| BuildValidator
BuildValidator -->|Integrity Check| BuildManifest
BuildManifest -->|Build Artifacts| ManifestBuild

%% Governance Defense Connections
GovValidator -->|Policy Check| GovPolicy
GovPolicy -->|Compliance State| GovAudit
GovAudit -->|Audit Manifest| ManifestRoot

%% Source Repository Connections
SrcRoot -->|Source Files| SrcInput
SrcInput -->|Security Layer| SrcCrypt
SrcCrypt -->|Access Control| SrcPriv
SrcPriv -->|Public Key Mgmt| SrcPub

%% Zero Trust Network Connections
ZTValidator -->|Zero Knowledge| PhantomEncoder
PhantomEncoder -->|Cryptographic Proof| CryptoProof
CryptoProof -->|State Validation| StateManager

%% Cross-Layer Defense Connections
CoreNode -.->|Zero Trust Protocol| ZTValidator
CLIGateway -.->|Command Trust Verification| ZTValidator
BuildOrchestrator -.->|Build Trust Verification| ZTValidator
GovValidator -.->|Governance Trust Verification| ZTValidator

%% Manifest Flow Connections
ManifestBuild -->|Validated Manifest| ManifestRoot
ManifestRoot -->|Public Distribution| ManifestPublic
ManifestRoot -->|Private Vault| ManifestPrivate

%% Source to Build Flow
SrcInput -->|Compilation Input| BuildOrchestrator
SrcCrypt -->|Security Validation| BuildValidator
SrcPub -->|Key Verification| CryptoProof
```

```
    classDef coreLayer fill:#ff6b6b,stroke:#333,stroke-width:3px
    classDef cliLayer fill:#4ecdc4,stroke:#333,stroke-width:3px
    classDef buildLayer fill:#45b7d1,stroke:#333,stroke-width:3px
    classDef govLayer fill:#f9ca24,stroke:#333,stroke-width:3px
    classDef srcLayer fill:#6c5ce7,stroke:#333,stroke-width:3px
    classDef manifestLayer fill:#a29bfe,stroke:#333,stroke-width:3px
    classDef zeroTrustLayer fill:#fd79a8,stroke:#333,stroke-width:3px

    class CoreNode,CoreValidator,CoreManifest coreLayer
    class CLIGateway,CLIValidator,CLIManifest cliLayer
    class BuildOrchestrator,BuildValidator,BuildManifest buildLayer
    class GovValidator,GovPolicy,GovAudit govLayer
    class SrcRoot,SrcInput,SrcCrypt,SrcPriv,SrcPub srcLayer
    class ManifestRoot,ManifestBuild,ManifestPublic,ManifestPrivate manifestLayer
    class ZTValidator,PhantomEncoder,CryptoProof,StateManager zeroTrustLayer
```

## Zero Trust Topology Implementation

### Defense Layer Architecture

The zero trust topology implements multiple defense layers that operate as autonomous protocol nodes within the OBINexus ecosystem. Each layer maintains independent validation capabilities while participating in the distributed verification network.

**Core Defense Layer** operates as the primary security boundary, implementing cryptographic validation and XML manifest control. The Core Protocol Node maintains system state integrity while coordinating with downstream validation systems.

**CLI Defense Layer** provides command-line interface security through sanitized command processing and CLI-specific manifest generation. The CLI Gateway Node ensures that all command-line interactions undergo comprehensive validation before system integration.

**Build Defense Layer** manages compilation integrity through orchestrated build processes and comprehensive artifact validation. The Build Orchestrator Node coordinates with source repository topology to ensure secure compilation workflows.

**Governance Defense Layer** implements policy enforcement and audit trail management through systematic compliance verification. The Governance Validator Node maintains regulatory compliance while enabling flexible operational procedures.

### Node-Network Verification Protocol

The verification protocol implements distributed validation across all topology nodes using the established Sinphasé methodology. Each node operates independently while contributing to system-wide security through coordinated verification procedures.

**Zero Trust Validation Network** provides cryptographic proof generation through the Phantom Encoder pattern and comprehensive state management. The Zero Trust Validator coordinates with all defense layers to ensure systematic security enforcement.

**Cryptographic Proof Engine** implements mathematical verification based on the odd perfect number cryptographic integrity framework. The proof engine generates verifiable certificates for each operational transition while maintaining zero-knowledge security properties.

**Gated State Manager** enforces state transition protocols that require explicit validation before system progression. The state manager implements checkpoint-based validation that ensures comprehensive quality assurance before deployment authorization.

## XML Manifest Flow Architecture

The manifest flow architecture implements systematic handling of XML manifests across source repositories and build artifacts. When public/private .xml files are deleted from src/ and missy/ directories, the system automatically relocates build/.xml manifests to root/src for continued operation.

**Manifest Management Topology** provides centralized coordination of manifest storage and distribution. The system maintains build/.xml artifacts in root/src directories while enabling continued CLI build operations through relocated manifest processing.

**Source Repository Integration** ensures that source files maintain security through layered cryptographic validation. The .in source files undergo validation through .crypt security files, .privation access control, and .pub public key management before build integration.

## CLI Integration with Governance

The CLI integration implements pre-governance validation that ensures sample CLI builds operate correctly before formal governance approval. The system supports github.com/gov-repo integration through standardized manifest validation procedures.

**Command Trust Verification** provides systematic validation of CLI commands through zero trust protocols before system integration. Each CLI operation undergoes cryptographic verification and policy compliance assessment before execution authorization.

**Build Integrity Validation** ensures that relocated manifest operations maintain system integrity while enabling flexible development workflows. The system validates build artifacts against source repositories through comprehensive cryptographic verification procedures.

## Implementation Requirements

The zero trust topology requires systematic implementation of defense layers with comprehensive verification procedures. Each node must implement both CLI and library interfaces while maintaining cost monitoring and phase-aware activation logic as specified in the Sinphasé framework.

The system implements automatic manifest relocation procedures that maintain build functionality when source .xml files are removed. CLI build operations continue through relocated manifest processing while maintaining comprehensive security validation throughout the compilation workflow.