Technical Architecture Specification

CyberGraph Intelligence Platform (CGIP)

Multi-Tenant Cybersecurity Consulting Platform

Version: 2.0
Date: October 26, 2025
Classification: Internal - Development
Document Type: Technical Architecture
Status: Approved

Executive Summary

This document defines the comprehensive technical architecture for the **CyberGraph Intelligence Platform (CGIP)**, a multi-tenant cybersecurity consulting platform built on **OpenSPG + KAG** for knowledge graph management, **Next.js** for frontend, **FastAPI** for backend services, **PostgreSQL** for application data, and **Neo4j Enterprise** for graph storage. The architecture emphasizes **data isolation**, **scalability**, **security**, and **extensibility** to support multiple consulting clients with shared threat intelligence.

1. Architecture Overview

1.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────────┐
│                    CLIENT LAYER (Multi-Tenant)                      │
│                                                                      │
│  ┌──────────────────┐  ┌──────────────────┐  ┌──────────────────┐        │
│  │ Client A UI  │  │ Client B UI  │  │ Consultant   │        │
│  │  (Next.js)   │  │  (Next.js)   │  │  Portal UI   │        │
│  └──────────────────┘  └──────────────────┘  └──────────────────┘        │
│                                                                      │
└─────────────────────────┬───────────────────────────────────────────┘
                          │ HTTPS/TLS 1.3
                          ▼
┌─────────────────────────────────────────────────────────────────────┐
│                    API GATEWAY / LOAD BALANCER                      │
│                     (Nginx / Kong / Traefik)                        │
└─────────────────────────┬───────────────────────────────────────────┘
                          │
            ┌─────────────┼─────────────────┐
            ▼             ▼                 ▼
┌──────────────────┐ ┌──────────────────┐ ┌──────────────────┐
│   FastAPI      │ │   FastAPI      │ │   FastAPI      │
│  Web Service   │ │  Agent Zero    │ │  Document      │
│   (Primary)    │ │     API        │ │  Generator     │
└────────┬─────────┘ └────────┬─────────┘ └────────┬─────────┘
```

```
          │                 │                  │
          └─────────────────────┼──────────────────────┘
                           │
          ┌────────────────────┴─────────────────────┐
          ▼                                  ▼
  ┌──────────────────────┐           ┌───────────────────────────┐
  │   PostgreSQL     │           │       Redis          │
  │  (Multi-Tenant)  │           │  (Caching/Session)   │
  │  Application DB  │           └───────────────────────────┘
  └──────────────────────┘
          │
          └──────────────────┐
                             ▼
  ┌──────────────────────────────────────────────────────────────────────────────────┐
  │              KNOWLEDGE GRAPH LAYER (OpenSPG + Neo4j)             │
  │  ┌─────────────────────────────────────────────────────────────────────────┐   │
  │  │                    OpenSPG Engine                            │   │
  │  │  ┌────────────────┐  ┌────────────────┐  ┌────────────────────┐       │   │
  │  │  │SPG-Schema   │  │SPG-Builder │  │SPG-Reasoner │       │   │
  │  │  └────────────────┘  └────────────────┘  └────────────────────┘       │   │
  │  └─────────────────────────────────────────────────────────────────────────┘   │
  │                                                                  │
  │  ┌─────────────────────────────────────────────────────────────────────────┐   │
  │  │              KAG Framework (Reasoning)                       │   │
  │  │  ┌────────────────┐  ┌────────────────┐  ┌────────────────────┐       │   │
  │  │  │KAG-Builder  │  │KAG-Solver  │  │ KAG-Model   │       │   │
  │  │  └────────────────┘  └────────────────┘  └────────────────────┘       │   │
  │  └─────────────────────────────────────────────────────────────────────────┘   │
  │                                                                  │
  │  ┌─────────────────────────────────────────────────────────────────────────┐   │
  │  │            OneKE (Knowledge Extraction)                      │   │
  │  │  ┌────────────────┐  ┌────────────────┐  ┌────────────────────┐       │   │
  │  │  │Schema Agent│  │Extract Agnt│  │Reflect Agnt │       │   │
  │  │  └────────────────┘  └────────────────┘  └────────────────────┘       │   │
  │  └─────────────────────────────────────────────────────────────────────────┘   │
  └────────────────────────────────┬─────────────────────────────────────────────────┘
                            │
          ┌──────────────────────┼──────────────────────┐
          ▼                  ▼                  ▼
  ┌────────────────────┐  ┌──────────────────┐  ┌──────────────────┐
  │  Neo4j (Shared)│  │Neo4j Client A│  │Neo4j Client B│
  │  Threat Intel  │  │    Graph     │  │    Graph     │
  │     Graph      │  │              │  │              │
  └────────────────────┘  └──────────────────┘  └──────────────────┘

          ┌──────────────────────┐
          │      Milvus      │
          │   Vector Store   │
          │   (Embeddings)   │
          └──────────────────────┘

  ┌──────────────────────────────────────────────────────────────────────────────────┐
  │                    INTEGRATION LAYER                             │
  │  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐         │
  │  │    Kafka     │  │  Agent Zero  │  │    SIEM      │             │
  │  │Event Stream  │  │Integration   │  │Connectors    │             │
```

```
│  └───────────────┘  └───────────────┘  └───────────────┘            │
└─────────────────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────────────────┐
│                    MONITORING &amp; OPERATIONS                        │
│  ┌───────────────┐  ┌───────────────┐  ┌───────────────┐            │
│  │ Prometheus  │  │   Grafana   │  │  ELK Stack  │            │
│  │  (Metrics)  │  │(Dashboards) │  │   (Logs)    │            │
│  └───────────────┘  └───────────────┘  └───────────────┘            │
└─────────────────────────────────────────────────────────────────────────┘
```

1.2 Architecture Principles

1. **Multi-Tenancy First**: All components designed for tenant isolation from day one
2. **Security by Default**: Encryption, authentication, authorization at every layer
3. **Scalability**: Horizontal scaling for all stateless services
4. **Observability**: Comprehensive monitoring, logging, and tracing
5. **Vendor Neutrality**: Avoid lock-in to specific cloud providers
6. **API-First**: All features exposed via well-documented APIs
7. **Event-Driven**: Asynchronous processing for long-running tasks
8. **Immutable Infrastructure**: Container-based deployment with infrastructure as code

2. Multi-Tenancy Architecture

2.1 Tenant Isolation Strategy

Approach: Hybrid Multi-Database + Row-Level Security

PostgreSQL (Application Data)

- **Strategy**: Schema-per-tenant with shared schemas
- **Implementation**:
  - Each tenant has dedicated schema: `tenant_&lt;tenant_id&gt;`
  - Shared schema for common data: `shared`
  - Row-level security (RLS) policies as backup
  - Middleware extracts `tenant_id` from JWT and sets search path

Schema Structure:

```
-- Tenant-specific schema
CREATE SCHEMA tenant_abc123;
CREATE TABLE tenant_abc123.assets (...);
CREATE TABLE tenant_abc123.assessments (...);
CREATE TABLE tenant_abc123.vulnerabilities (...);
```

```
-- Shared schema
CREATE SCHEMA shared;
CREATE TABLE shared.cve_database (...);
CREATE TABLE shared.mitre_attack (...);
CREATE TABLE shared.iec62443_requirements (...);
```

Neo4j (Knowledge Graph)

- **Strategy**: Multi-database (one database per tenant + shared)
- **Implementation**:
  - Neo4j Enterprise supports multiple databases
  - Each tenant: `neo4j_tenant_&lt;tenant_id&gt;`
  - Shared database: `neo4j_shared` (CVE, MITRE, standards)
  - Connection pooling per database
  - Queries scoped to tenant database

Database Structure:

```
// Tenant-specific database
CREATE DATABASE neo4j_tenant_abc123;

// Shared database (read-only for tenants)
CREATE DATABASE neo4j_shared;

// Cross-database query (when needed)
USE neo4j_tenant_abc123;
CALL apoc.cypher.run('USE neo4j_shared RETURN ...');
```

2.2 Tenant Context Management

JWT Token Structure:

```
{
  "sub": "user_id_12345",
  "tenant_id": "abc123",
  "roles": ["consultant", "iec62443_assessor"],
  "client_ids": ["abc123", "def456"],
  "exp": 1698345600,
  "iat": 1698259200
}
```

Middleware Flow:

```
# FastAPI middleware for tenant context
from fastapi import Request, HTTPException
from jose import jwt

async def tenant_middleware(request: Request, call_next):
    # Extract JWT from Authorization header
    token = request.headers.get("Authorization", "").replace("Bearer ", "")

    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
        tenant_id = payload.get("tenant_id")

        if not tenant_id:
            raise HTTPException(status_code=401, detail="Missing tenant_id")

        # Set tenant context in request state
        request.state.tenant_id = tenant_id
        request.state.user_id = payload.get("sub")
        request.state.roles = payload.get("roles", [])

        # Set PostgreSQL search path
        await set_search_path(f"tenant_{tenant_id}, shared")

        # Set Neo4j database context
        request.state.neo4j_db = f"neo4j_tenant_{tenant_id}"

    except jwt.JWTError:
        raise HTTPException(status_code=401, detail="Invalid token")

    response = await call_next(request)
    return response
```

2.3 Tenant Provisioning

Automated Tenant Onboarding:

1. Create tenant record in PostgreSQL `tenants` table
2. Create PostgreSQL schema: `tenant_&lt;tenant_id&gt;`
3. Run schema migrations for tenant
4. Create Neo4j database: `neo4j_tenant_&lt;tenant_id&gt;`
5. Initialize tenant-specific graph schema (UCO + ICS ontology)
6. Create default admin user for tenant
7. Generate API keys for integrations
8. Send welcome email with credentials

Provisioning API:

```
POST /api/v1/admin/tenants
{
  "name": "Acme Manufacturing Inc",
  "domain": "acme.com",
  "admin_email": "admin@acme.com",
  "plan": "professional",
  "settings": {
    "max_consultants": 10,
    "max_assets": 5000,
    "enable_agent_zero": true
  }
}
```

3. Frontend Architecture

3.1 Next.js Application Structure

Project Structure:

```
cgip-frontend/
├── app/                           # Next.js 14 App Router
│   ├── (auth)/                    # Auth routes group
│   │   ├── login/
│   │   ├── logout/
│   │   └── sso/
│   ├── (dashboard)/               # Main dashboard group
│   │   ├── layout.tsx             # Dashboard layout with sidebar
│   │   ├── page.tsx               # Dashboard home
│   │   ├── assets/                # Asset management
│   │   ├── vulnerabilities/       # Vulnerability management
│   │   ├── assessments/           # IEC 62443 assessments
│   │   ├── pentests/              # Penetration testing
│   │   ├── threats/               # Threat modeling
│   │   ├── reports/               # Document generation
│   │   └── settings/              # Client settings
│   ├── api/                       # Next.js API routes (BFF)
│   │   ├── auth/
│   │   ├── proxy/                 # Proxy to FastAPI
│   │   └── webhooks/
│   └── layout.tsx                 # Root layout
├── components/                    # React components
│   ├── ui/                        # shadcn/ui components
│   ├── charts/                    # Chart components (Recharts)
│   ├── graph/                     # Graph visualization (D3.js)
│   └── forms/                     # Form components
├── lib/                           # Utilities
│   ├── api.ts                     # API client
│   ├── auth.ts                    # Auth utilities
│   └── utils.ts                   # Helper functions
├── hooks/                         # Custom React hooks
├── stores/                        # Zustand stores
├── types/                         # TypeScript types
└── styles/                        # Global styles
```

3.2 State Management

Zustand Stores:

```
// stores/tenantStore.ts
interface TenantState {
  currentTenant: Tenant | null;
  tenants: Tenant[];
  setCurrentTenant: (tenant: Tenant) =&gt; void;
  loadTenants: () =&gt; Promise&lt;void&gt;;
}

export const useTenantStore = create&lt;TenantState&gt;((set) =&gt; ({
  currentTenant: null,
  tenants: [],
  setCurrentTenant: (tenant) =&gt; set({ currentTenant: tenant }),
  loadTenants: async () =&gt; {
    const tenants = await fetchTenants();
    set({ tenants });
  },
}));
```

React Query for Server State:

```
// hooks/useAssets.ts
import { useQuery, useMutation } from '@tanstack/react-query';

export const useAssets = (tenantId: string) =&gt; {
  return useQuery({
    queryKey: ['assets', tenantId],
    queryFn: () =&gt; fetchAssets(tenantId),
    staleTime: 5 * 60 * 1000, // 5 minutes
  });
};

export const useCreateAsset = () =&gt; {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: (asset: NewAsset) =&gt; createAsset(asset),
    onSuccess: () =&gt; {
      queryClient.invalidateQueries({ queryKey: ['assets'] });
    },
  });
};
```

3.3 UI Component Library (shadcn/ui)

Key Components:

- `&lt;Button&gt;`, `&lt;Input&gt;`, `&lt;Select&gt;`, `&lt;Checkbox&gt;`, `&lt;RadioGroup&gt;`
- `&lt;Dialog&gt;`, `&lt;Popover&gt;`, `&lt;Tooltip&gt;`, `&lt;DropdownMenu&gt;`

- `&lt;Table&gt;`, `&lt;Card&gt;`, `&lt;Tabs&gt;`, `&lt;Accordion&gt;`
- `&lt;Form&gt;` (React Hook Form integration)
- `&lt;Toast&gt;` (Sonner notifications)
- `&lt;Command&gt;` (Command palette)

Styling with Tailwind CSS:

```
// components/AssetCard.tsx
export function AssetCard({ asset }: { asset: Asset }) {
  return (
    &lt;Card className="w-full"&gt;
      &lt;CardHeader&gt;
        &lt;CardTitle className="flex items-center justify-between"&gt;
          <span>{asset.name}</span>
          &lt;Badge variant={asset.critical ? "destructive" : "default"}&gt;
            {asset.type}
          &lt;/Badge&gt;
        &lt;/CardTitle&gt;
        &lt;CardDescription&gt;{asset.description}&lt;/CardDescription&gt;
      &lt;/CardHeader&gt;
      &lt;CardContent&gt;
        <div>
          <div>
            <span>CVEs:</span>
            <span>{asset.cve_count}</span>
          </div>
          <div>
            <span>Risk Score:</span>
            &lt;RiskBadge score={asset.risk_score} /&gt;
          </div>
        </div>
      &lt;/CardContent&gt;
      &lt;CardFooter&gt;
        &lt;Button variant="outline" size="sm" onClick={() =&gt; viewDetails(asset)}&gt;
          View Details
        &lt;/Button&gt;
      &lt;/CardFooter&gt;
    &lt;/Card&gt;
  );
}
```

3.4 Graph Visualization

D3.js Force-Directed Graph:

```
// components/graph/KnowledgeGraphViewer.tsx
import * as d3 from 'd3';

export function KnowledgeGraphViewer({ data }: { data: GraphData }) {
  const svgRef = useRef&lt;SVGSVGElement&gt;(null);

  useEffect(() =&gt; {
    if (!svgRef.current) return;

    const svg = d3.select(svgRef.current);
    const width = svgRef.current.clientWidth;
    const height = svgRef.current.clientHeight;

    // Force simulation
    const simulation = d3.forceSimulation(data.nodes)
      .force('link', d3.forceLink(data.links).id((d: any) =&gt; d.id))
      .force('charge', d3.forceManyBody().strength(-300))
      .force('center', d3.forceCenter(width / 2, height / 2))
      .force('collision', d3.forceCollide().radius(30));

    // Render nodes and links
    // ... (detailed D3 code omitted for brevity)

  }, [data]);

  return &lt;svg ref={svgRef} className="w-full h-full" /&gt;;
}
```

4. Backend Architecture

4.1 FastAPI Application Structure

Project Structure:

```
cgip-api/
├── app/
│   ├── main.py                   # FastAPI app entry point
│   ├── core/
│   │   ├── config.py             # Configuration (Pydantic Settings)
│   │   ├── security.py           # JWT, encryption, auth
│   │   ├── dependencies.py       # Dependency injection
│   │   └── middleware.py         # Custom middleware
│   ├── api/
│   │   ├── v1/
│   │   │   ├── __init__.py
│   │   │   ├── auth.py           # Authentication endpoints
│   │   │   ├── tenants.py        # Tenant management
│   │   │   ├── assets.py         # Asset management
│   │   │   ├── vulnerabilities.py
│   │   │   ├── assessments.py    # IEC 62443 assessments
│   │   │   ├── pentests.py       # Penetration testing
│   │   │   ├── threats.py        # Threat modeling
│   │   │   ├── agent_zero.py     # Agent Zero API
│   │   │   ├── graph.py          # Knowledge graph queries
│   │   │   ├── reports.py        # Document generation
│   │   │   └── integrations.py   # External integrations
│   │   └── deps.py               # Route dependencies
│   ├── models/
│   │   ├── database.py           # SQLAlchemy models
│   │   ├── graph.py              # Graph entity models
│   │   └── schemas.py            # Pydantic schemas
│   ├── services/
│   │   ├── openspg_service.py    # OpenSPG integration
│   │   ├── neo4j_service.py      # Neo4j operations
│   │   ├── oneke_service.py      # OneKE extraction
│   │   ├── kag_service.py        # KAG reasoning
│   │   ├── agent_zero_service.py # Agent Zero coordination
│   │   ├── report_service.py     # Document generation
│   │   └── risk_service.py       # Risk calculation
│   ├── repositories/
│   │   ├── asset_repository.py
│   │   ├── vulnerability_repository.py
│   │   └── assessment_repository.py
│   ├── tasks/
│   │   ├── ingestion_tasks.py    # Celery tasks for CVE ingestion
│   │   ├── analysis_tasks.py     # Background analysis
│   │   └── report_tasks.py       # Report generation
│   └── utils/
│       ├── graph_utils.py
│       ├── security_utils.py
│       └── validation_utils.py
├── tests/
├── alembic/                      # Database migrations
├── Dockerfile
├── requirements.txt
└── pyproject.toml
```

4.2 Dependency Injection

```
# app/api/deps.py
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from jose import jwt, JWTError
from sqlalchemy.orm import Session

security = HTTPBearer()

def get_db() -&gt; Session:
    \"\"\"Get database session\"\"\"
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db: Session = Depends(get_db)
) -&gt; User:
    \"\"\"Get current authenticated user\"\"\"
    try:
        payload = jwt.decode(
            credentials.credentials,
            settings.SECRET_KEY,
            algorithms=[settings.ALGORITHM]
        )
        user_id = payload.get("sub")
        if user_id is None:
            raise HTTPException(status_code=401, detail="Invalid token")
    except JWTError:
        raise HTTPException(status_code=401, detail="Invalid token")

    user = db.query(User).filter(User.id == user_id).first()
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")

    return user

def get_current_tenant(
    credentials: HTTPAuthorizationCredentials = Depends(security)
) -&gt; str:
    \"\"\"Extract tenant_id from JWT\"\"\"
    try:
        payload = jwt.decode(
            credentials.credentials,
            settings.SECRET_KEY,
            algorithms=[settings.ALGORITHM]
        )
        tenant_id = payload.get("tenant_id")
        if tenant_id is None:
            raise HTTPException(status_code=401, detail="Missing tenant_id")
        return tenant_id
    except JWTError:
        raise HTTPException(status_code=401, detail="Invalid token")

def get_neo4j_session(tenant_id: str = Depends(get_current_tenant)):
    \"\"\"Get Neo4j session scoped to tenant\"\"\"
    driver = Neo4jDriver()
    session = driver.session(database=f"neo4j_tenant_{tenant_id}")
    try:
        yield session
    finally:
        session.close()
```

4.3 API Endpoint Example

```
# app/api/v1/assessments.py
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List

router = APIRouter()

@router.post("/assessments", response_model=AssessmentResponse)
async def create_assessment(
    assessment: AssessmentCreate,
    tenant_id: str = Depends(get_current_tenant),
    db: Session = Depends(get_db),
    neo4j: Session = Depends(get_neo4j_session),
    current_user: User = Depends(get_current_user)
```

```
):
    \"\"\"
    Create new IEC 62443 assessment for client
    \"\"\"
    # Validate tenant access
    if not current_user.has_tenant_access(tenant_id):
        raise HTTPException(status_code=403, detail="Access denied")

    # Create assessment record
    db_assessment = Assessment(
        tenant_id=tenant_id,
        name=assessment.name,
        standard="IEC 62443",
        created_by=current_user.id
    )
    db.add(db_assessment)
    db.commit()
    db.refresh(db_assessment)

    # Initialize assessment in knowledge graph
    assessment_node = await create_assessment_node(
        neo4j,
        assessment_id=db_assessment.id,
        tenant_id=tenant_id
    )

    # Trigger background analysis
    analyze_iec62443_gaps.delay(db_assessment.id, tenant_id)

    return db_assessment

@router.get("/assessments/{assessment_id}", response_model=AssessmentDetail)
async def get_assessment(
    assessment_id: int,
    tenant_id: str = Depends(get_current_tenant),
    db: Session = Depends(get_db),
    neo4j: Session = Depends(get_neo4j_session)
):
    \"\"\"
    Get assessment details with gap analysis results
    \"\"\"
    assessment = db.query(Assessment).filter(
        Assessment.id == assessment_id,
        Assessment.tenant_id == tenant_id
    ).first()

    if not assessment:
        raise HTTPException(status_code=404, detail="Assessment not found")

    # Fetch gap analysis from knowledge graph
    gaps = await get_assessment_gaps(neo4j, assessment_id)

    return {
        **assessment.__dict__,
        "gaps": gaps,
        "coverage": calculate_coverage(gaps)
    }
```

5. Knowledge Graph Architecture

5.1 OpenSPG Integration

SPG-Schema Definition:

```
# app/schemas/spg_schema.py
from openspg import Schema, EntityType, RelationType, Property

# Define ICS ontology
class ICSSchema(Schema):
    # Entity types
    Asset = EntityType(
        name="Asset",
        properties=[
            Property("name", "String"),
            Property("type", "String"),  # PLC, HMI, SCADA, etc.
            Property("manufacturer", "String"),
            Property("model", "String"),
            Property("firmware_version", "String"),
            Property("ip_address", "String"),
            Property("zone_id", "String"),
            Property("security_level", "Integer"),  # IEC 62443 SL 1-4
            Property("criticality", "String"),  # Low, Medium, High, Critical
        ]
    )

    Vulnerability = EntityType(
        name="Vulnerability",
        properties=[
            Property("cve_id", "String"),
            Property("cvss_score", "Float"),
            Property("severity", "String"),
            Property("description", "Text"),
            Property("exploitability", "Float"),
            Property("public_exploit_available", "Boolean"),
        ]
    )

    Zone = EntityType(
        name="Zone",
        properties=[
            Property("name", "String"),
            Property("security_level", "Integer"),  # Target SL
            Property("purpose", "String"),
            Property("criticality", "String"),
        ]
    )

    Control = EntityType(
        name="Control",
        properties=[
            Property("control_id", "String"),  # FR-1, SR 1.1, etc.
            Property("name", "String"),
            Property("description", "Text"),
            Property("iec62443_ref", "String"),
            Property("implemented", "Boolean"),
            Property("evidence", "Text"),
        ]
    )

    # Relationship types
    AFFECTS = RelationType(
        name="AFFECTS",
        source="Vulnerability",
        target="Asset",
        properties=[
            Property("affected_version", "String"),
            Property("patched_version", "String"),
        ]
    )

    LOCATED_IN = RelationType(
        name="LOCATED_IN",
        source="Asset",
        target="Zone"
    )

    IMPLEMENTS = RelationType(
        name="IMPLEMENTS",
        source="Zone",
        target="Control"
    )

    MITIGATES = RelationType(
        name="MITIGATES",
        source="Control",
        target="Vulnerability"
    )

    CONNECTS_TO = RelationType(
        name="CONNECTS_TO",
        source="Asset",
        target="Asset",
        properties=[
            Property("protocol", "String"),
            Property("port", "Integer"),
            Property("conduit_id", "String"),
        ]
    )
```

5.2 KAG Framework Integration

KAG-Solver for Multi-Hop Reasoning:

```
# app/services/kag_service.py
from kag import KAGSolver

class KAGReasoningService:
    def __init__(self, neo4j_session, openspg_client):
        self.solver = KAGSolver(neo4j_session, openspg_client)

    async def find_attack_paths(
        self,
        source_asset_id: str,
        target_asset_id: str,
        attacker_profile: str = "external"
    ) -&gt; List[AttackPath]:
        \"\"\"
        Find attack paths from source to target using KAG reasoning
        \"\"\"
        # Define query in KGDSL
        query = f\"\"\"
        MATCH path = (source:Asset {{id: '{source_asset_id}'}})-[*1..10]-&gt;(target:Asse
        WHERE all(r IN relationships(path) WHERE
          type(r) IN ['CONNECTS_TO', 'EXPLOITS', 'COMPROMISES', 'ACCESSES']
        )
        WITH path,
          [r IN relationships(path) | r.difficulty] AS difficulties,
          [r IN relationships(path) | r.stealth] AS stealth_scores
        RETURN path,
          reduce(s = 1, d IN difficulties | s * d) AS path_difficulty,
          reduce(s = 1, st IN stealth_scores | s * st) AS path_stealth
        ORDER BY path_difficulty ASC, path_stealth DESC
        LIMIT 10
        \"\"\"

        # Execute via KAG-Solver (handles logical form decomposition)
        results = await self.solver.execute(query)

        # Post-process results
        attack_paths = []
        for result in results:
            path = self._parse_attack_path(result['path'])
            path.difficulty = result['path_difficulty']
            path.stealth = result['path_stealth']
            path.steps = self._extract_attack_steps(result['path'])
            attack_paths.append(path)

        return attack_paths

    async def calculate_risk_propagation(
        self,
        vulnerability_id: str,
        tenant_id: str
    ) -&gt; RiskPropagationResult:
        \"\"\"
        Calculate cascading risk impact of vulnerability
        \"\"\"
        query = f\"\"\"
        MATCH (vuln:Vulnerability {{id: '{vulnerability_id}'}})-[:AFFECTS]-&gt;(asset:Ass
        OPTIONAL MATCH (asset)-[:CONNECTS_TO*1..5]-&gt;(downstream:Asset)
        WITH vuln, asset, collect(DISTINCT downstream) AS affected_assets
        RETURN vuln,
          asset.criticality AS direct_criticality,
          [a IN affected_assets | a.criticality] AS cascade_criticality,
          vuln.cvss_score AS base_risk
        \"\"\"

        result = await self.solver.execute(query)

        # Calculate aggregated risk
        direct_impact = self._calculate_impact(result['direct_criticality'])
        cascade_impact = sum([
            self._calculate_impact(c) * 0.7  # Decay factor
            for c in result['cascade_criticality']
        ])

        total_risk = result['base_risk'] * (direct_impact + cascade_impact)

        return RiskPropagationResult(
            vulnerability_id=vulnerability_id,
            base_risk=result['base_risk'],
            direct_impact=direct_impact,
            cascade_impact=cascade_impact,
            total_risk=total_risk,
            affected_assets=len(result['cascade_criticality']) + 1
        )
```

5.3 OneKE Knowledge Extraction

Document Extraction Service:

```
# app/services/oneke_service.py
from oneke import OneKEClient

class KnowledgeExtractionService:
    def __init__(self):
        self.oneke = OneKEClient(api_url=settings.ONEKE_URL)

    async def extract_from_pdf(
        self,
        pdf_path: str,
        schema_type: str = "ics_topology"
    ) -&gt; ExtractionResult:
        \"\"\"
        Extract structured knowledge from PDF using OneKE
        \"\"\"
        # Define extraction schema
        if schema_type == "ics_topology":
            schema = {
                "entities": [
                    {"type": "Device", "attributes": ["name", "type", "ip_address"]},
                    {"type": "Network", "attributes": ["name", "subnet", "vlan"]},
                    {"type": "Zone", "attributes": ["name", "security_level"]}
                ],
                "relationships": [
                    {"type": "CONNECTS_TO", "source": "Device", "target": "Device"},
                    {"type": "LOCATED_IN", "source": "Device", "target": "Zone"}
                ]
            }

        # Call OneKE extraction
        result = await self.oneke.extract(
            file_path=pdf_path,
            schema=schema,
            language="en",
            domain="cybersecurity"
        )

        # Validate and enrich results
        validated_entities = self._validate_entities(result['entities'])
        validated_relationships = self._validate_relationships(result['relationships'])

        return ExtractionResult(
            entities=validated_entities,
            relationships=validated_relationships,
            confidence=result['confidence'],
            extraction_time=result['processing_time']
        )

    def _validate_entities(self, entities: List[Dict]) -&gt; List[Entity]:
        \"\"\"Validate and enrich extracted entities\"\"\"
        validated = []
        for entity in entities:
            # Check confidence threshold
            if entity['confidence'] &lt; 0.7:
                entity['requires_review'] = True

            # Normalize device types
            if entity['type'] == 'Device':
                entity['normalized_type'] = self._normalize_device_type(
                    entity['attributes']['type']
                )

            validated.append(entity)

        return validated
```

6. Agent Zero Integration

6.1 Agent Zero API Endpoints

```
# app/api/v1/agent_zero.py
from fastapi import APIRouter, Depends, BackgroundTasks
from app.services.agent_zero_service import AgentZeroService

router = APIRouter(prefix="/agent-zero", tags=["agent-zero"])

@router.get("/query")
async def query_knowledge_graph(
    query: str,
    tenant_id: str = Depends(get_current_tenant),
    neo4j: Session = Depends(get_neo4j_session)
):
    \"\"\"
    Agent Zero queries knowledge graph for infrastructure data

    **Performance Target**: p95 &lt; 500ms
    \"\"\"
    service = AgentZeroService(neo4j)

    result = await service.execute_query(
        query=query,
        tenant_id=tenant_id,
        timeout_ms=500
    )

    return {
        "nodes": result.nodes,
        "relationships": result.relationships,
        "execution_time_ms": result.execution_time
    }

@router.post("/update")
async def update_knowledge_graph(
    update: KnowledgeGraphUpdate,
    tenant_id: str = Depends(get_current_tenant),
    neo4j: Session = Depends(get_neo4j_session),
    current_user: User = Depends(get_current_user)
):
    \"\"\"
    Agent Zero updates knowledge graph with discovered vulnerabilities

    **Performance Target**: p95 &lt; 200ms
    \"\"\"
    service = AgentZeroService(neo4j)

    # Validate update against schema
    validation_result = await service.validate_update(update)
    if not validation_result.valid:
        raise HTTPException(status_code=400, detail=validation_result.errors)

    # Apply update with audit trail
    result = await service.apply_update(
        update=update,
        tenant_id=tenant_id,
        source="agent_zero",
        user_id=current_user.id
    )

    return {
        "success": True,
        "nodes_created": result.nodes_created,
        "relationships_created": result.relationships_created,
        "audit_id": result.audit_id
    }

@router.post("/select-path")
async def select_attack_path(
    request: AttackPathRequest,
    tenant_id: str = Depends(get_current_tenant),
    neo4j: Session = Depends(get_neo4j_session)
):
    \"\"\"
    Get optimal attack path recommendations for Agent Zero

    **Performance Target**: &lt; 1 second for complex paths
    \"\"\"
    service = AgentZeroService(neo4j)

    paths = await service.find_attack_paths(
        source_asset_id=request.source_asset_id,
        target_asset_id=request.target_asset_id,
        attacker_profile=request.attacker_profile,
        max_hops=request.max_hops or 10
    )

    # Rank paths by success probability
    ranked_paths = await service.rank_paths(
        paths=paths,
        criteria={
            "difficulty": 0.4,
            "stealth": 0.3,
            "impact": 0.3
        }
    )

    return {
        "paths": ranked_paths,
        "recommended_path_id": ranked_paths[0].id if ranked_paths else None
    }

@router.get("/exploits")
async def get_exploits(
    cve_id: Optional[str] = None,
    capec_id: Optional[str] = None,
    technique_id: Optional[str] = None,
    neo4j: Session = Depends(get_neo4j_session)
):
    \"\"\"
    Get matching exploits from database
    **Performance Target**: p95 &lt; 300ms
    \"\"\"
    service = AgentZeroService(neo4j)

    exploits = await service.find_exploits(
        cve_id=cve_id,
        capec_id=capec_id,
        mitre_technique_id=technique_id
    )

    return {
        "exploits": exploits,
        "count": len(exploits)
    }

@router.post("/simulate")
async def simulate_attack(
    simulation: AttackSimulation,
    background_tasks: BackgroundTasks,
    tenant_id: str = Depends(get_current_tenant)
):
    \"\"\"
    Run attack simulation in sandbox environment
    \"\"\"
    service = AgentZeroService()

    # Queue simulation as background task
    task_id = await service.queue_simulation(
        simulation=simulation,
        tenant_id=tenant_id
    )

    return {
        "task_id": task_id,
        "status": "queued",
        "estimated_time_seconds": simulation.estimated_duration
    }

@router.post("/feedback")
async def submit_feedback(
    feedback: LearningFeedback,
    tenant_id: str = Depends(get_current_tenant)
):
    \"\"\"
    Submit learning feedback to improve path selection

    **Performance Target**: p95 &lt; 100ms
    \"\"\"
    service = AgentZeroService()

    await service.record_feedback(
        path_id=feedback.path_id,
        outcome=feedback.outcome,
        execution_time=feedback.execution_time,
        stealth_score=feedback.stealth_score,
        notes=feedback.notes
    )

    # Trigger model retraining if threshold reached
    await service.check_retrain_threshold()

    return {"success": True}
```

6.2 Agent Zero Simulation Orchestration

```
# app/services/agent_zero_service.py
class AgentZeroService:
    async def queue_simulation(
        self,
        simulation: AttackSimulation,
        tenant_id: str
    ) -&gt; str:
        \"\"\"Queue attack simulation as Celery task\"\"\"
        task = run_attack_simulation.delay(
            tenant_id=tenant_id,
            path_id=simulation.path_id,
            exploits=simulation.exploits,
            target_asset_id=simulation.target_asset_id,
            sandbox_config=simulation.sandbox_config
        )

        return task.id

    async def rank_paths(
        self,
        paths: List[AttackPath],
        criteria: Dict[str, float]
    ) -&gt; List[AttackPath]:
        \"\"\"
        Rank attack paths using ML model trained on historical success
        \"\"\"
        # Load trained model
        model = self._load_ranking_model()

        # Extract features for each path
        features = []
        for path in paths:
            features.append({
                "hop_count": len(path.steps),
                "avg_cvss": np.mean([s.cvss_score for s in path.steps]),
                "public_exploits": sum([s.public_exploit for s in path.steps]),
                "avg_difficulty": np.mean([s.difficulty for s in path.steps]),
                "avg_stealth": np.mean([s.stealth for s in path.steps]),
                "target_criticality": path.target_asset.criticality
            })

        # Predict success probability
        X = pd.DataFrame(features)
        probabilities = model.predict_proba(X)[:, 1]

        # Apply criteria weights and rank
        for i, path in enumerate(paths):
            path.success_probability = probabilities[i]
            path.weighted_score = (
                criteria["difficulty"] * (1 - path.avg_difficulty) +
                criteria["stealth"] * path.avg_stealth +
                criteria["impact"] * path.target_criticality
            ) * path.success_probability

        # Sort by weighted score
        ranked = sorted(paths, key=lambda p: p.weighted_score, reverse=True)

        return ranked
```

7. Document Generation

7.1 Template Engine

```
# app/services/report_service.py
from jinja2 import Environment, FileSystemLoader
from docxtpl import DocxTemplate
from weasyprint import HTML

class ReportGenerationService:
    def __init__(self):
        self.jinja_env = Environment(loader=FileSystemLoader("templates/reports"))

    async def generate_iec62443_gap_analysis(
        self,
        assessment_id: int,
        tenant_id: str,
        neo4j: Session
    ) -&gt; bytes:
        \"\"\"Generate IEC 62443 gap analysis report\"\"\"
        # Query knowledge graph for assessment data
        data = await self._get_assessment_data(neo4j, assessment_id)

        # Load template
        doc = DocxTemplate("templates/reports/iec62443_gap_analysis.docx")

        # Prepare context
        context = {
            "client_name": data["client_name"],
            "assessment_date": datetime.now().strftime("%Y-%m-%d"),
            "zones": data["zones"],
            "gaps": data["gaps"],
            "coverage_summary": data["coverage_summary"],
            "recommendations": data["recommendations"]
        }

        # Render template
        doc.render(context)

        # Save to bytes
        output = io.BytesIO()
        doc.save(output)
        output.seek(0)

        return output.getvalue()

    async def generate_pentest_report(
        self,
        pentest_id: int,
        tenant_id: str
    ) -&gt; bytes:
        \"\"\"Generate penetration test report with findings\"\"\"
        # Get pentest results from database
        pentest = await self._get_pentest_results(pentest_id, tenant_id)

        # Render HTML from Jinja2 template
        template = self.jinja_env.get_template("pentest_report.html")
        html_content = template.render(pentest=pentest)

        # Convert HTML to PDF using WeasyPrint
        pdf = HTML(string=html_content).write_pdf()

        return pdf
```

7.2 Template Structure (IEC 62443)

```
{# templates/reports/iec62443_gap_analysis.html #}

&lt;html&gt;
&lt;head&gt;
  &lt;title&gt;IEC 62443 Gap Analysis Report&lt;/title&gt;
  &lt;style&gt;
    /* Report styling */
  &lt;/style&gt;
&lt;/head&gt;
&lt;body&gt;
  <h1>IEC 62443 Gap Analysis Report</h1>
  <p><strong>Client:</strong> {{ client_name }}</p>
  <p><strong>Assessment Date:</strong> {{ assessment_date }}</p>

  <h2>Executive Summary</h2>
  <p>
    This report presents the results of an IEC 62443 security assessment
    for {{ client_name }}'s industrial control system environment.
  </p>

  <h3>Coverage Summary</h3>
  &lt;table&gt;
    &lt;tr&gt;
      &lt;th&gt;Security Level&lt;/th&gt;
      &lt;th&gt;Required Controls&lt;/th&gt;
      &lt;th&gt;Implemented&lt;/th&gt;
      &lt;th&gt;Coverage %&lt;/th&gt;
    &lt;/tr&gt;
    {% for sl in coverage_summary %}
    &lt;tr&gt;
      &lt;td&gt;SL-{{ sl.level }}&lt;/td&gt;
      &lt;td&gt;{{ sl.required }}&lt;/td&gt;
      &lt;td&gt;{{ sl.implemented }}&lt;/td&gt;
      &lt;td&gt;{{ sl.coverage }}%&lt;/td&gt;
    &lt;/tr&gt;
    {% endfor %}
  &lt;/table&gt;

  <h2>Gap Analysis by Zone</h2>
  {% for zone in zones %}
  <h3>Zone: {{ zone.name }} (Target SL-{{ zone.security_level }})</h3>

  <h4>Missing Controls</h4>
  <ul>
    {% for gap in zone.gaps %}
    <li>
      <strong>{{ gap.control_id }}</strong>: {{ gap.name }}
      <p>{{ gap.description }}</p>
      <p><em>Remediation Effort:</em> {{ gap.effort }}</p>
    </li>
    {% endfor %}
  </ul>
  {% endfor %}

  <h2>Recommendations</h2>
  <ol>
    {% for rec in recommendations %}
    <li>
      <strong>{{ rec.priority }}</strong>: {{ rec.text }}
      <p>{{ rec.justification }}</p>
    </li>
    {% endfor %}
  </ol>
&lt;/body&gt;
&lt;/html&gt;
```

8. Deployment Architecture

8.1 Docker Compose (Development)

```
# docker-compose.yml
version: '3.9'

services:
  # Frontend
  frontend:
    build:
      context: ./cgip-frontend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    environment:
      - NEXT_PUBLIC_API_URL=http://api:8000
    depends_on:
      - api

  # Backend API
  api:
    build:
      context: ./cgip-api
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://cgip:password@postgres:5432/cgip
      - NEO4J_URI=bolt://neo4j:7687
      - REDIS_URL=redis://redis:6379
      - OPENSPG_URL=http://openspg:8887
      - ONEKE_URL=http://oneke:8080
      - KAG_URL=http://kag:8888
    depends_on:
      - postgres
      - neo4j
      - redis
      - openspg

  # PostgreSQL
  postgres:
    image: postgres:16-alpine
    environment:
      - POSTGRES_DB=cgip
      - POSTGRES_USER=cgip
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  # Neo4j
  neo4j:
    image: neo4j:5.23-enterprise
    environment:
      - NEO4J_AUTH=neo4j/password
      - NEO4J_ACCEPT_LICENSE_AGREEMENT=yes
      - NEO4J_dbms_memory_heap_max__size=4G
    volumes:
      - neo4j_data:/data
      - neo4j_logs:/logs
    ports:
      - "7474:7474"
      - "7687:7687"

  # OpenSPG
  openspg:
    image: openspg/openspg:0.5.0
    environment:
      - NEO4J_URI=bolt://neo4j:7687
      - NEO4J_USER=neo4j
      - NEO4J_PASSWORD=password
    ports:
      - "8887:8887"
    depends_on:
      - neo4j

  # OneKE
  oneke:
    image: openspg/oneke:1.0
    environment:
      - MODEL_PATH=/models
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    volumes:
      - oneke_models:/models
    ports:
      - "8080:8080"

  # KAG Framework
  kag:
    image: openspg/kag:0.4
    environment:
      - OPENSPG_URL=http://openspg:8887
      - NEO4J_URI=bolt://neo4j:7687
    ports:
      - "8888:8888"
    depends_on:
      - openspg

  # Redis
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  # Milvus (Vector DB)
  milvus:
    image: milvusdb/milvus:2.4.0
    environment:
      - ETCD_ENDPOINTS=etcd:2379
    ports:
      - "19530:19530"
    depends_on:
      - etcd

  etcd:
    image: quay.io/coreos/etcd:v3.5.0
    environment:
      - ETCD_AUTO_COMPACTION_MODE=revision
      - ETCD_AUTO_COMPACTION_RETENTION=1000
    volumes:
      - etcd_data:/etcd

  # Kafka
  kafka:
    image: confluentinc/cp-kafka:7.6.0
    environment:
      - KAFKA_BROKER_ID=1
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
      - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
    ports:
      - "9092:9092"
    depends_on:
      - zookeeper

  zookeeper:
    image: confluentinc/cp-zookeeper:7.6.0
    environment:
      - ZOOKEEPER_CLIENT_PORT=2181
    volumes:
      - zookeeper_data:/var/lib/zookeeper

  # Celery Worker
  celery_worker:
    build:
      context: ./cgip-api
      dockerfile: Dockerfile
    command: celery -A app.tasks worker --loglevel=info
    environment:
      - DATABASE_URL=postgresql://cgip:password@postgres:5432/cgip
      - REDIS_URL=redis://redis:6379
    depends_on:
      - postgres
      - redis
      - kafka

  # Nginx
  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - frontend
      - api

  # Prometheus
  prometheus:
    image: prom/prometheus:latest
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus_data:/prometheus

  # Grafana
  grafana:
    image: grafana/grafana:latest
    ports:
      - "3001:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana
    depends_on:
      - prometheus

volumes:
  postgres_data:
  neo4j_data:
  neo4j_logs:
  redis_data:
  oneke_models:
  etcd_data:
  zookeeper_data:
  prometheus_data:
  grafana_data:
```

8.2 Kubernetes Deployment (Production)

```
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cgip-api
  namespace: cgip
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cgip-api
  template:
    metadata:
      labels:
        app: cgip-api
    spec:
      containers:
      - name: cgip-api
        image: cgip/api:latest
        ports:
        - containerPort: 8000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: cgip-secrets
              key: database-url
        - name: NEO4J_URI
          value: "bolt://neo4j-service:7687"
        resources:
          requests:
            memory: "2Gi"
            cpu: "1000m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: cgip-api-service
  namespace: cgip
spec:
  selector:
    app: cgip-api
  ports:
  - port: 80
    targetPort: 8000
  type: LoadBalancer
```

9. Security Architecture

9.1 Authentication and Authorization

JWT Token Generation:

```
# app/core/security.py
from jose import jwt
from datetime import datetime, timedelta

def create_access_token(
    user_id: str,
    tenant_id: str,
    roles: List[str],
    expires_delta: timedelta = timedelta(hours=8)
) -&gt; str:
    \"\"\"Create JWT access token\"\"\"
    expire = datetime.utcnow() + expires_delta

    payload = {
        "sub": user_id,
        "tenant_id": tenant_id,
        "roles": roles,
        "exp": expire,
        "iat": datetime.utcnow()
    }

    token = jwt.encode(payload, settings.SECRET_KEY, algorithm="HS256")
    return token
```

Role-Based Access Control:

```
# app/core/permissions.py
from enum import Enum

class Permission(str, Enum):
    READ_ASSETS = "read:assets"
    WRITE_ASSETS = "write:assets"
    READ_ASSESSMENTS = "read:assessments"
    WRITE_ASSESSMENTS = "write:assessments"
    RUN_PENTESTS = "run:pentests"
    MANAGE_USERS = "manage:users"
    ADMIN = "admin"

ROLE_PERMISSIONS = {
    "viewer": [Permission.READ_ASSETS, Permission.READ_ASSESSMENTS],
    "consultant": [Permission.READ_ASSETS, Permission.WRITE_ASSETS,
                   Permission.READ_ASSESSMENTS, Permission.WRITE_ASSESSMENTS],
    "pentest_lead": [Permission.READ_ASSETS, Permission.RUN_PENTESTS],
    "admin": [Permission.ADMIN]  # Grants all permissions
}

def has_permission(user_roles: List[str], required_permission: Permission) -&gt; bool:
    \"\"\"Check if user has required permission\"\"\"
    for role in user_roles:
        if Permission.ADMIN in ROLE_PERMISSIONS.get(role, []):
            return True
        if required_permission in ROLE_PERMISSIONS.get(role, []):
            return True
    return False
```

9.2 Data Encryption

Encryption at Rest:

- PostgreSQL: Transparent Data Encryption (TDE) via `pgcrypto`
- Neo4j: Encrypted storage with enterprise edition
- File storage: AES-256 encryption for all uploaded files

Encryption in Transit:

- TLS 1.3 for all API communications
- mTLS for inter-service communication in production
- Certificate management via Let's Encrypt / cert-manager

10. Monitoring and Observability

10.1 Metrics Collection

Prometheus Metrics:

```
# app/core/metrics.py
from prometheus_client import Counter, Histogram, Gauge

# API metrics
api_requests_total = Counter(
    'cgip_api_requests_total',
    'Total API requests',
    ['method', 'endpoint', 'status']
)

api_request_duration = Histogram(
    'cgip_api_request_duration_seconds',
    'API request duration',
    ['method', 'endpoint']
)

# Knowledge graph metrics
kg_query_duration = Histogram(
    'cgip_kg_query_duration_seconds',
    'Knowledge graph query duration',
    ['tenant_id', 'query_type']
)

kg_nodes_total = Gauge(
    'cgip_kg_nodes_total',
    'Total nodes in knowledge graph',
    ['tenant_id']
)

# Agent Zero metrics
agent_zero_simulations = Counter(
    'cgip_agent_zero_simulations_total',
    'Agent Zero simulations',
    ['tenant_id', 'status']
)
```

10.2 Logging

Structured Logging:

```
# app/core/logging.py
import structlog

logger = structlog.get_logger()

# Usage
logger.info(
    "assessment_created",
    tenant_id=tenant_id,
    assessment_id=assessment_id,
    user_id=user_id,
    assessment_type="IEC62443"
)
```

10.3 Distributed Tracing

OpenTelemetry Integration:

```
# app/core/tracing.py
from opentelemetry import trace
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor

tracer = trace.get_tracer(__name__)

# Instrument FastAPI
FastAPIInstrumentor.instrument_app(app)

# Manual spans
with tracer.start_as_current_span("kg_query"):
    result = await execute_kg_query(query)
```

Appendices

Appendix A: API Performance Targets

Endpoint Category	p50	p90	p95	p99
Authentication	50ms	100ms	150ms	300ms
Asset Management	100ms	200ms	300ms	500ms
KG Queries (simple)	200ms	500ms	1s	2s
KG Queries (complex)	1s	2s	3s	5s
Agent Zero API	200ms	400ms	500ms	1s
Report Generation	5s	10s	15s	30s

Appendix B: Database Sizing

Component	Small (<10 clients)	Medium (10-50)	Large (50+)
PostgreSQL	100GB	500GB	2TB
Neo4j (per tenant)	10GB	50GB	200GB
Neo4j (shared)	50GB	100GB	500GB
Milvus	20GB	100GB	500GB

Appendix C: Ontology References

- **UCO**: https://github.com/Ebiquity/Unified-Cybersecurity-Ontology
- **MITRE ATT&CK STIX**: https://documentation.eccenca.com/23.3/build/tutorial-how-to-link-ids-to-osint/lift-data-from-STIX-2.1-data-of-mitre-attack/
- **IEC 62443 Ontology**: Knowledge-based Engineering of Automation Systems using Ontologies (Glawe, Tebbe, Fay, Niemann)

Document Approval

Role	Name	Signature	Date
Chief Architect			
Technical Lead			
DevOps Lead			
Security Lead			

This document is confidential and proprietary. Distribution restricted to authorized personnel.