KGCS: Building a Frozen Knowledge Graph for Expert Cybersecurity AI

Version: 1.0
Date: January 2026
Purpose: Design, architecture, and implementation of a standards-backed cybersecurity knowledge graph that enables safe, explainable, and hallucination-free AI reasoning about vulnerabilities, attacks, defenses, and threat intelligence.

Executive Summary

KGCS (Cybersecurity Knowledge Graph) is a **frozen, immutable, standards-aligned ontology** that integrates 9 MITRE security taxonomies (CVE, CWE, CPE, CVSS, CAPEC, ATT&CK, D3FEND, CAR, SHIELD, ENGAGE) to create a single source of truth for cybersecurity AI systems.

Why This Matters

Current cybersecurity AI systems suffer from:

- **Semantic drift:** Standards are reinterpreted to fit models, losing original meaning
- **Hallucination:** LLMs invent causal links that don't exist in source data
- **Opacity:** Answers can't be traced back to authoritative sources
- **Fragility:** New standard releases break downstream systems
- **Scope creep:** Adding one feature (e.g., incidents) requires redesigning the entire graph

KGCS solves this by enforcing three invariants:

1. **Authoritative alignment:** Every ontology class maps 1:1 to official JSON/STIX schemas
2. **Explicit provenance:** Every relationship is traceable to source data
3. **Layered architecture:** Core (frozen facts) + Extensions (contextual, temporal, inferred)

The result is an AI that can reason confidently, explain its reasoning, and remain maintainable as standards evolve.

Table of Contents

- 1. Core Architecture
- 2. The Vulnerability Causality Chain

- 3. Defense, Detection, and Deception Coverage
- 4. Engagement and Strategic Reasoning
- 5. Standards Alignment by Layer
- 6. Modular Ontology Design
- 7. RAG-Safe Traversal Paths
- 8. Extension Layers (Incident, Risk, Threat Actor)
- 9. Implementation Roadmap
- 10. Why This Architecture Scales
- 11. Security Guarantees
- 12. Limitations and Future Work
- 13. For the AI Engineer
- Conclusion

---

1. Core Architecture

1.1 Layered Ontology Design

KGCS is built on **four immutable layers**, each with a clear semantic boundary:

```
┌──────────────────────────────────────────────────────────────┐
│ LAYER 4: EXTENSION LAYER (Contextual, Temporal, Inferred)    │
├──────────────────────────────────────────────────────────────┤
│ • Incident Extension: ObservedTechnique, DetectionEvent      │
│ • Risk Extension: RiskAssessment, RiskScenario               │
│ • ThreatActor Extension: Attribution, GroupActivity          │
│ (Never modifies or overrides Core semantics)                 │
└──────────────────────────────────────────────────────────────┘

         ▲                                     ▲
         │ instantiates / observes / reasons   │
         │ (but does not alter Core)          │
         │                                     │

┌──────────────────────────────────────────────────────────────┐
│ LAYER 3: BRIDGE LAYER (Integrated Core Ontology)             │
├──────────────────────────────────────────────────────────────┤
│ Imports all standards + establishes causal chain:             │
│ CPE → CVE → CWE → CAPEC → ATT&CK → D3FEND/CAR/SHIELD/ENGAGE│
│ (Frozen; immutable once published)                           │
└──────────────────────────────────────────────────────────────┘

         ▲         ▲          ▲          ▲         ▲
         │ imports │ imports  │ imports  │ imports │
         │         │          │          │         │
```

```
  ┌─────┴────┐   ┌───┴───┐  ┌──┴──┐  ┌────┴────┐  ┌───┴───┐
  │   CPE    │   │  CVE  │  │ CWE │  │  CAPEC  │  │ ATT&CK │
  │Ontology  │   │Ontology│ │Ontology│ Ontology │  │Ontology│
  └─────┬────┘   └───┬───┘  └──┬──┘  └────┬────┘  └───┬───┘
        │            │          │          │           │
  ┌─────┴────────────┴──────────┴──────────┴───────────┴──────┐
  │       LAYER 2: MODULAR STANDARD ONTOLOGIES                 │
  ├─────────────────────────────────────────────────────────────┤
  │ Each ontology = independent, reusable RDF/OWL module:       │
  │ • No circular imports                                       │
  │ • 1:1 alignment with JSON/STIX schemas                      │
  │ • Standalone use case support                               │
  │ (E.g., asset inventory uses CPE only; vuln mgmt uses CPE+CVE)
  └─────────────────────────────────────────────────────────────┘

        ▲         ▲          ▲          ▲          ▲
        │ models  │ models   │ models   │ models   │
        │         │          │          │          │
  ┌─────┴────┐  ┌───┴────┐  ┌──┴──┐  ┌────┴────┐  ┌───┴───┐
  │  NVD     │  │  NVD   │  │MITRE│  │  MITRE  │  │ MITRE │
  │ CPE JSON │  │CVE JSON│  │CWE  │  │ CAPEC   │  │ATT&CK │
  │          │  │+ CVSS  │  │XML  │  │  JSON   │  │ STIX  │
  └─────┬────┘  └───┬────┘  └──┬──┘  └────┬────┘  └───┬───┘
        │           │          │          │           │
  ┌─────┴───────────┴──────────┴──────────┴───────────┴───────┐
  │   LAYER 1: AUTHORITATIVE EXTERNAL STANDARDS                 │
  ├─────────────────────────────────────────────────────────────┤
  │ • NVD (NIST): CPE, CVE, CVSS specifications                 │
  │ • MITRE: CWE, CAPEC, ATT&CK, D3FEND, CAR, SHIELD, ENGAGE   │
  │ • Official JSON / STIX / XML schemas (versioned)            │
  │ (Sources of truth; never modified)                         │
  └─────────────────────────────────────────────────────────────┘
```

1.2 Core Principle: External Standards = Authoritative Truth

Rule 1: Every ontology class maps 1:1 to a JSON object or field group in official standards.

Rule 2: Every relationship is explicitly traceable to source data.

Rule 3: No invented semantics. The ontology is a **lens**, not a replacement.

---

2. The Vulnerability Causality Chain

The **canonical path** through the knowledge graph is:

```
CPE (Asset Identifier)
  ↓ affected_by
CVE (Vulnerability Disclosure)
  ├─ scored_by → CVSS (Severity Assessment)
  └─ caused_by
CWE (Weakness Root Cause)
  ↓ exploited_by
CAPEC (Attack Pattern Abstraction)
```

```
  ├─ implements_as / maps_to
ATT&CK Technique (Adversary Behavior)
  ├─ mitigated_by → D3FEND (Defensive Technique)
  ├─ detected_by → CAR (Detection Analytic)
  ├─ countered_by → SHIELD (Deception Technique)
  └─ disrupted_by → ENGAGE (Strategic Concept)
```

2.1 Why This Chain Matters

This path **enforces explainability**. Every step:

- Exists in authoritative JSON/STIX
- Has a documented source
- Is independently verifiable
- Cannot be skipped without losing information

Example Question: "How does CVE-2025-1234 relate to ATT&CK Technique T1059?"

Answer (with provenance):

```
CVE-2025-1234
  → caused_by: CWE-94 (Improper Control of Generation of Code)
      [source: cvejson.weaknesses[]]
  → exploited_by: CAPEC-242 (Code Injection)
      [source: capec JSON Related_Weaknesses]
  → maps_to: ATT&CK T1059 (Command and Scripting Interpreter)
      [source: MITRE-documented mapping]
```

Every hop is verifiable.

2.2 Critical Invariant: Vulnerabilities Affect Configurations, Not Platforms

Not this:

```
CVE → Platform (wrong)
```

But this:

```
CVE → PlatformConfiguration
      (version bounds, update level, criteria)
      ↓
      includes → Platform
```

Why? A platform (e.g., "Windows 10") is not vulnerable. A *specific configuration* of that platform is (e.g., "Windows 10 version 21H2 without KB5012234").

NVD encodes this as logical expressions in `configurations[].nodes[].cpeMatch[]`. We preserve that.

---

3. Defense, Detection, and Deception Coverage

3.1 Parallel Control Dimensions

Three independent dimensions answer three questions about ATT&CK techniques:

Dimension	Question	Standard	Semantics
Mitigation	How to prevent or reduce impact?	D3FEND	Defensive technique
Detection	How to detect execution?	CAR	Detection analytic + data sources
Deception	How to deceive or manipulate?	SHIELD	Deception technique

These are **not hierarchical**. All three can apply to the same technique.

3.2 Example: T1087 (Account Discovery)

```
T1087 (ATT&CK)
├─ mitigated_by → D3-ID-001 (Credential Access Restriction)
├─ mitigated_by → D3-OT-002 (Harden User Account Privilege)
├─ detected_by → CAR-2016-12-001 (Create Remote Process)
├─ detected_by → CAR-2013-10-002 (SMB Write Request)
└─ countered_by → SHIELD-002 (Fake Account Indicators)
```

No coverage is perfect. This graph lets an SOC:

- See all available defenses at a glance
- Choose which to implement based on cost/effectiveness
- Detect when gaps exist
- Reason about layered defense

---

4. Engagement and Strategic Reasoning

4.1 ENGAGE Does Not Map to Techniques

Unlike D3FEND, CAR, and SHIELD, ENGAGE operates at a **strategic level**, not a technical one.

```
ENGAGE Concept
├─ targets → Threat Actor Group
├─ informs → Response Strategy
└─ influences → when/how D3FEND/CAR/SHIELD are deployed
```

ENGAGE answers:

- How do we interact with adversaries?
- What are their decision points?
- How do we disrupt their operations at a macro level?

4.2 Example: Attribution as Engagement

```
ENGAGE Strategy: "Coordinated Public Attribution"
├─ targets: Nation-state actors
├─ objective: Increase operational costs
└─ informs: Decision to deploy SHIELD (deception)
    to gather intelligence for attribution
```

---

5. Standards Alignment by Layer

5.1 Layer 1: CPE (Platform Identification)

Source: NVD CPE Dictionary JSON 2.3

Ontology Class: `Platform (CPE)`

Key Properties:

- `cpeUri` (canonical string)
- `part` , `vendor` , `product` , `version` , `update` , `edition` , ...
- `deprecated` (boolean)

Edges:

- `Platform ──deprecates──> Platform` (obsolescence tracking)
- `Platform ──isVariantOf──> Platform` (version relationships)

Use Case: "What software/hardware is in our infrastructure?"

5.2 Layer 2: CVE + CVSS (Vulnerability Disclosure)

Source: NVD CVE API 2.0 + CVSS JSON

Ontology Classes:

- `Vulnerability (CVE)`
- `VulnerabilityScore (CVSS)` (versioned: v2, v3.1, v4.0)

- `Reference` (provenance)

Key Properties:

- `cveId`, `description`, `published`, `lastModified`
- `vulnStatus` (Analyzed, Undergoing Analysis, Deferred, Disputed, Rejected)
- `cisaDateAdded`, `cisaActionDue` (federal remediation mandates)

Critical:

- Each CVSS version = **separate node** (never overwrite)
- `affected_by` relationship goes to `PlatformConfiguration`, not `Platform`

Use Case: "Is our system vulnerable to CVE-2025-1234?"

5.3 Layer 3: CWE (Weakness Root Cause)

Source: MITRE CWE JSON

Ontology Class: `Weakness (CWE)`

Key Properties:

- `cweId`, `name`, `abstraction` (Pillar, Class, Base, Variant)
- `description`, `status`

Edges:

- `Weakness ──parent_of──> Weakness` (hierarchical)
- `Weakness ──member_of──> WeaknessView`
- `Weakness ──exploited_by──> AttackPattern`

Use Case: "What underlying flaw does this CVE exploit?"

5.4 Layer 4: CAPEC (Attack Pattern Abstraction)

Source: MITRE CAPEC JSON

Ontology Class: `AttackPattern (CAPEC)`

Key Properties:

- `capecId`, `name`, `description`
- `likelihood`, `severity`, `prerequisites`

Edges:

- `AttackPattern —exploits—> Weakness` (weakness → behavior bridge)
- `AttackPattern —related_to—> AttackPattern`
- `AttackPattern —maps_to—> Technique` (to ATT&CK)

Use Case: "How are weaknesses exploited in practice?"

5.5 Layer 5: ATT&CK (Adversary Tradecraft)

Source: MITRE ATT&CK STIX 2.1

Ontology Classes:

- `Technique` , `SubTechnique` (attack-pattern type)
- `Tactic` (x-mitre-tactic)
- `Group` , `Software` (actor / campaign context)
- `DataSource` , `DataComponent` (detection metadata)

Key Relationships:

- `Technique —part_of—> Tactic`
- `SubTechnique —subtechnique_of—> Technique` (functional)
- `Group —uses—> Technique`
- `Technique —detected_by—> DataComponent`

Use Case: "What are adversaries actually doing?"

5.6 Layer 6: D3FEND (Defensive Techniques)

Source: MITRE D3FEND STIX

Ontology Class: `DefensiveTechnique (D3FEND)`

Subtypes:

- `DetectionTechnique` (how to detect)
- `DenialTechnique` (how to block)
- `DisruptionTechnique` (how to interfere)

Key Properties:

- `d3fendId` , `name` , `sophisticationLevel` , `costLevel` , `scope`
- `implementationStatus` (Proposed, Beta, Stable, Deprecated)

Edges:

- `DefensiveTechnique —mitigates—> Technique` (direct)
- `DefensiveTechnique —mitigates—> Weakness` (root cause)
- `DefensiveTechnique —weakens—> AttackPattern`

Use Case: "How do we defend against this technique?"

5.7 Layer 7: CAR (Detection Analytics)

Source: MITRE CAR JSON/YAML

Ontology Class: `DetectionAnalytic (CAR)`

Key Properties:

- `carId` , `name` , `description`
- `techniques[]` (which ATT&CK techniques it detects)
- `data_sources[]` (what data it needs)

Edges:

- `DetectionAnalytic —detects—> Technique`
- `DetectionAnalytic —requires—> DataSource`

Use Case: "How do we detect this behavior?"

5.8 Layer 8: SHIELD (Deception Techniques)

Source: MITRE SHIELD STIX (experimental)

Ontology Class: `DeceptionTechnique (SHIELD)`

Subtypes:

- `HoneypotDeception` (attract and deceive)
- `MisdirectionDeception` (misdirect)
- `InformationDecoy` (false data)
- `SocialManipulation` (social engineering)

Key Properties:

- `shieldId`, `name`, `targetAdversary`
- `sophisticationLevel`, `deploymentComplexity`
- `primaryObjective` (Detect, Disrupt, Deceive, Gather Intelligence)

Edges:

- `DeceptionTechnique ——counters——> Technique`
- `DeceptionTechnique ——reveals——> Technique` (expose TTPs)

Use Case: "How do we deceive or observe adversaries?"

5.9 Layer 9: ENGAGE (Strategic Engagement)

Source: MITRE ENGAGE Framework (conceptual)

Ontology Class: `EngagementConcept (ENGAGE)`

Subtypes:

- `DisruptionStrategy` (disrupt operations)
- `OperationalInterference` (interfere with C2)
- `CapabilityDegradation` (weaken capabilities)
- `AttributionStrategy` (identify and attribute)
- `ResilienceBuilding` (strengthen defenses)

Key Properties:

- `engageId`, `name`, `strategyType`
- `timeframe` (Immediate, Short-term, Medium-term, Long-term)
- `operationalLevel` (Tactical, Operational, Strategic)
- `riskLevel`, `legalConsiderations`

Edges:

- `EngagementConcept ——disrupts——> Technique`
- `EngagementConcept ——targets——> Group`
- `EngagementConcept ——informs——> ResponseStrategy`

Use Case: "How do we interact strategically with adversaries?"

6. Modular Ontology Design

6.1 Why Separate Ontologies?

Instead of one monolithic ontology, KGCS uses **modular, independent OWL files**:

```
docs/ontology/owl/
├── cpe-ontology-v1.0.owl          (standalone)
├── cve-ontology-v1.0.owl          (imports CPE)
├── cwe-ontology-v1.0.owl          (standalone)
├── capec-ontology-v1.0.owl        (imports CWE)
├── attck-ontology-v1.0.owl        (standalone)
├── d3fend-ontology-v1.0.owl       (imports Core)
├── car-ontology-v1.0.owl          (imports Core)
├── shield-ontology-v1.0.owl       (imports Core)
├── engage-ontology-v1.0.owl       (imports Core)
├── core-ontology-extended-v1.0.owl (imports all above)
└── [extensions]
    ├── incident-ontology-extension-v1.0.owl
    ├── risk-ontology-extension-v1.0.owl
    └── threatactor-ontology-extension-v1.0.owl
```

6.2 Benefits of Modularity

Benefit	Enables
Reusability	Asset inventory systems use CPE alone; vulnerability management uses CPE+CVE
Independent Versioning	CPE 3.0 can be added without breaking CVE/CWE/etc.
Scope Limiting	Teams work on one standard without touching others
Testing	Each ontology validated independently before integration
Future Growth	New standards (CVSS-NG, D3FEND extensions) slot in cleanly

6.3 Import Graph (No Circular Dependencies)

```
Core Ontology
├─ imports: cpe-ontology-v1.0.owl
├─ imports: cve-ontology-v1.0.owl (imports CPE)
├─ imports: cwe-ontology-v1.0.owl
├─ imports: capec-ontology-v1.0.owl (imports CWE)
├─ imports: attck-ontology-v1.0.owl
├─ imports: d3fend-ontology-v1.0.owl
├─ imports: car-ontology-v1.0.owl
├─ imports: shield-ontology-v1.0.owl
```

```
└─ imports: engage-ontology-v1.0.owl

Extensions (never in Core; always reference Core)
├─ incident-ontology-extension-v1.0.owl (imports Core)
├─ risk-ontology-extension-v1.0.owl (imports Core)
└─ threatactor-ontology-extension-v1.0.owl (imports Core)
```

Critical Rule: Extensions **reference** Core, never the reverse.

---

7. RAG-Safe Traversal Paths

7.1 Pre-Approved Query Templates

To prevent RAG hallucination, KGCS defines **approved traversal templates**. These enforce:

1. No layer skipping
2. No circular reasoning without context
3. Only authoritative edges

Example Template T-CORE-01: "How does this CVE impact this platform?"

```
Query: CVE-X → Platform-Y?
├─ CVE-X ──affects──> PlatformConfiguration-Z
│   (source: cvejson.configurations[].cpeMatch[].vulnerable)
├─ PlatformConfiguration-Z ──includes──> Platform-Y
│   (source: cpematch_api_json_2.0.schema)
└─ Result: Yes, if configuration matches
```

Example Template T-CORE-02: "What are all defenses against this technique?"

```
Query: Defenses for Technique-T?
├─ Technique-T ──mitigated_by──> DefensiveTechnique
│   (source: MITRE D3FEND stix relationships)
├─ Technique-T ──detected_by──> DetectionAnalytic
│   (source: CAR techniques[])
└─ Technique-T ──countered_by──> DeceptionTechnique
    (source: SHIELD mappings)
```

7.2 Forbidden Traversals

These are explicitly disallowed:

Forbidden	Why
CVE → ATT&CK (direct)	Must pass through CWE→CAPEC
Platform → CWE (direct)	Must pass through CVE

Forbidden	Why
Weakness → Group (direct)	No direct relationship; contextual only
Technique → Risk Score (direct)	Risk is extension-layer only

---

8. Extension Layers (Incident, Risk, Threat Actor)

8.1 Why Extensions Exist

Core ontology captures **what is known to be true** (facts from standards).

Extensions capture **what we observe or infer** (facts from operations/analysis).

Critical: Extensions **never modify Core classes**. They reference them.

8.2 Incident Extension

Purpose: Track temporal, contextual observations from SIEM/SOAR

Key Classes:

- `ObservedTechnique` (timestamp, confidence, evidence)
- `DetectionEvent` (alert, rule, sensor)
- `IncidentTimeline` (temporal sequence)

Relationship to Core:

```
ObservedTechnique ──instantiates──> Technique (ATT&CK)
DetectionEvent ──detects──> ObservedTechnique
IncidentTimeline ──includes──> DetectionEvent
```

Properties:

- `timestamp`, `firstSeen`, `lastSeen`, `confidence` (LOW|MEDIUM|HIGH)
- `sourceSystem`, `evidenceIds[]`

8.3 Risk Extension

Purpose: Capture risk assessment, prioritization, and decision-making

Key Classes:

- `RiskAssessment` (scenario + score)
- `RiskScenario` (vulnerability → impact)

- `RemediationDecision` (ACCEPT|MITIGATE|TRANSFER|AVOID)

Relationship to Core:

```
RiskScenario
├─ involves──> Vulnerability (CVE)
├─ affects──> Asset
├─ exploits──> AttackPattern (CAPEC)
└─ mitigated_by──> DefensiveTechnique (D3FEND)
```

Properties:

- `riskScore` (0-100), `likelihood`, `impact`
- `decision`, `decisionRationale`, `decisionDate`
- `owner`, `owner_contact`

8.4 ThreatActor Extension

Purpose: Track attribution claims with confidence levels

Key Classes:

- `AttributionClaim` (claim about group responsibility)
- `ThreatActorObservation` (observed behavior)

Relationship to Core:

```
AttributionClaim
├─ attributes_to──> Group (ATT&CK)
├─ based_on──> ObservedTechnique
├─ confidence──> (LOW|MEDIUM|HIGH|VERY_HIGH)
└─ evidenceIds[]→ ThreatActorObservation
```

Properties:

- `claimId`, `timestamp`, `confidence`
- `sourceSystem` (intel feed, SOAR, analyst), `analyst_note`

---

9. Implementation Roadmap

Phase 1: Core Standards (Complete ✅)

- ☑ CPE Ontology (v1.0)
- ☑ CVE Ontology (v1.0)

- [x] CWE Ontology (v1.0)
- [x] CAPEC Ontology (v1.0)
- [x] ATT&CK Ontology (v1.0)
- [x] D3FEND Ontology (v1.0)
- [x] CAR Ontology (v1.0)
- [x] SHIELD Ontology (v1.0)
- [x] ENGAGE Ontology (v1.0)
- [x] Core Ontology Extended (v1.0, imports all above)

Deliverable: 10 OWL/Turtle files with full 1:1 alignment to JSON/STIX schemas.

Phase 2: SHACL Validation (Next)

- [ ] Create SHACL shapes for each ontology
- [ ] Define constraint profiles: SOC (observational), EXEC (decision-focused), AI (hallucination-safe)
- [ ] Integrate pre-ingest validation (reject non-conforming data)
- [ ] Add runtime validation gates

Deliverable: SHACL shape files + validation pipeline

Phase 3: Data Ingestion (Planned)

- [ ] ETL scripts to load NVD CVE/CPE JSON
- [ ] MITRE STIX 2.1 parser for ATT&CK / D3FEND / SHIELD
- [ ] CWE / CAPEC / CAR JSON loaders
- [ ] Neo4j or RDF graph database setup
- [ ] Re-ingestion safety tests (versioning, conflict detection)

Deliverable: Production-ready data pipeline

Phase 4: Extension Layers (Planned)

- [ ] Incident Ontology Extension + temporal reasoning
- [ ] Risk Ontology Extension + prioritization models
- [ ] ThreatActor Ontology Extension + attribution logic

- [ ] RAG Traversal Template library

Deliverable: Safe, extensible query framework

Phase 5: AI Integration (Planned)

- [ ] RAG retrieval layer (safe traversals only)
- [ ] Explanation generation (path-based)
- [ ] Confidence scoring (traceability to source)
- [ ] Fine-tuning LLMs on KGCS queries

Deliverable: Hallucination-free cybersecurity AI

---

10. Why This Architecture Scales

10.1 Handling Standards Evolution

Scenario: CVSS 5.0 is released tomorrow.

Old approach: Rewrite vulnerability model, re-ingest all CVEs, hope nothing breaks.

KGCS approach:

1. Add `CVSSv50Score` subclass to `VulnerabilityScore` in CVE ontology
2. Load new NVD data with v5.0 metrics
3. Existing CVSS v3.1 / v4.0 nodes remain unchanged
4. RAG queries automatically see both versions

Why this works: Different CVSS versions are separate nodes with `scored_by` edges.

10.2 Adding New Standards

Scenario: You want to add NIST SP 800-53 (security controls).

KGCS approach:

1. Create `controls-ontology-v1.0.owl` (new, standalone)
2. Define edges: `DefensiveTechnique ──implements──> Control`
3. Import into Core Ontology
4. Existing queries unaffected; new queries enabled

Why this works: Modular design + no inheritance = clean extension.

10.3 Supporting Organizational Context

Scenario: You have 10,000 assets to integrate.

KGCS approach:

1. Assets live in **Asset Extension**, not Core
2. Core remains pristine (no CMDB data)
3. `Asset ──configured_with──> PlatformConfiguration` (from Core)
4. `Asset ──affected_by──> Vulnerability` (transitive)
5. Organizational changes don't break the graph

Why this works: Clear separation of authoritative (Core) vs. contextual (Extension).

10.4 Versioning and Rollback

Scenario: You ingest bad data into CAR, want to rollback one version.

KGCS approach:

1. Every entity has `cveId` , `capecId` , `carId` (stable external ID)
2. Nodes are immutable; only edges change
3. Version control at the RDF level:
```
# Before: CAR-2013-10-002 ──detects──> T1007 (with 3 evidence trails)
# Problem found: 1 trail is wrong
# After: Remove bad edge, keep 2 valid edges
# Rollback: Replay transaction, done
```

Why this works: Every relationship is traceable to source + timestamped.

---

11. Security Guarantees

11.1 No Hallucination

Every answer is a **path through nodes and edges**, each backed by:

- Stable external ID (CVE-YYYY-XXXX, T1234, etc.)
- Source document (JSON field, STIX property)
- Timestamp (when data was ingested)

11.2 Explainability

Every query result includes:

```json
{
  "answer": "CVE-2025-1234 relates to T1059",
  "confidence": "HIGH (authoritative path)",
  "path": [
    {
      "node": "CVE-2025-1234",
      "source": "NVD JSON id field"
    },
    {
      "edge": "caused_by",
      "source": "NVD JSON weaknesses[].description[]"
    },
    ...
  ],
  "evidence_urls": ["https://nvd.nist.gov/vuln/detail/CVE-2025-1234", ...]
}
```

11.3 Auditability

- Every node change is timestamped
- Every edge change is logged with provenance
- Data can be replayed from standard sources
- No hidden inferences

11.4 Compliance

- CVSS, CWE, CAPEC, ATT&CK mappings certified by MITRE
- CPE, CVE, CVSS alignment certified by NVD
- No custom interpretations
- Suitable for:
  - Regulatory reporting (PCI-DSS, NIST CSF, ISO 27001)
  - SOC playbooks and runbooks
  - Executive decision-making

---

12. Limitations and Future Work

12.1 Current Scope

KGCS 1.0 covers:

✅ Core standards (CVE, CWE, CPE, CVSS, CAPEC, ATT&CK, D3FEND, CAR, SHIELD, ENGAGE)
✅ 1:1 alignment with JSON/STIX schemas
✅ Modular OWL ontologies
✅ Formal semantics + constraints
🔄 SHACL validation (in progress)
🔄 Data ingestion pipeline (planned)
🔄 RAG integration (planned)

12.2 Intentional Exclusions

These are **contextual, not authoritative**, so they belong in Extensions:

- Threat actor motivations (organizational intel)
- Incident timelines (operational data)
- Asset inventory (CMDB)
- Risk scores (business decision)
- Detection rules (SOC implementation)
- Exploit code (ethical concerns)

12.3 Future Enhancements

1. **Formal Verification:** Prove consistency of ontology using theorem provers
2. **Temporal Reasoning:** Track when relationships were discovered/confirmed
3. **Uncertainty Quantification:** Bayesian inference over relationship weights
4. **Fine-Grained Provenance:** Link individual CVSS metrics to scoring rationale
5. **Multi-Language Support:** Translate ontology labels/descriptions to non-English
6. **Linked Data Publishing:** Expose as SPARQL endpoints for external systems

---

13. For the AI Engineer

13.1 Using KGCS for RAG

```
# Query: "What techniques can compromise this system?"

1. Extract asset from user context
   Asset → PlatformConfiguration → CVE

2. For each CVE:
   CVE → CWE → CAPEC → ATT&CK Technique
```

```
3. For each Technique:
   Technique → D3FEND (defenses)
   Technique → CAR (detection)
   Technique → SHIELD (deception)

4. Filter by:
   - Applicable to this environment (platform)
   - Confidence level (HIGH for authoritative paths)
   - Available data sources (CAR)

5. Return with full provenance:
   [
     {
       "technique": "T1234",
       "path": [...],
       "defenses": [...],
       "detection": [...],
       "confidence": "HIGH"
     }
   ]
```

13.2 Fine-Tuning LLMs

Use KGCS to:

1. **Ground training data** in authoritative sources
2. **Filter hallucinations** (reject answers not in path)
3. **Score confidence** (authoritative path = HIGH)
4. **Generate explanations** (path = explanation)

Conclusion

KGCS is built for one purpose: Enable AI systems to reason confidently about cybersecurity.

By aligning ontologies 1:1 with standards, enforcing explicit provenance, and layering contextual extensions cleanly, KGCS ensures that:

- ✅ Every answer is traceable to source
- ✅ Standards evolve without breaking the graph
- ✅ New contexts (assets, incidents, risks) integrate cleanly
- ✅ AI can explain its reasoning with evidence
- ✅ Humans retain control and auditability

The result is an expert cybersecurity AI that is trustworthy, maintainable, and compliant.

For more information:

- See `docs/ontology/` for formal ontology specifications
- See `.github/copilot-instructions.md` for AI agent governance
- See `docs/draft/` for detailed design documents