SBOM Analysis

Transitive Dependencies & Supply Chain Lineage

Cybersecurity Use Cases: Full Code-to-Cloud Traceability with Neo4j

Based on: pedroleitao-neo4j/cyber-sbom

© Neo4j 2024

What is an SBOM?

The "Pre-packaged Meal" Analogy

Think of an application as a pre-packaged meal:

- The **label** lists the ingredients (libraries/components).
- Developers don't write everything from scratch; they use libraries.
- Those libraries use *other* libraries, creating a **hidden chain**.

The Goal: Move from guessing risk to total traceability.

© Neo4j 2024

The Problem: Hidden Ingredients

Traditional tools focus on top-level dependencies, missing Transitive Risks.

- **Invisible Chains:** Risks buried 4-5 layers deep in the software lineage.
- **Zero-Day Lag:** Finding affected apps manually can take weeks.
- **The Gap:** Disconnect between Development (code) and Operations (cloud servers).

© Neo4j 2024

The Solution: Cyber-SBOM Graph

Building on the VPEM (Vulnerability Prioritization) model, this solution provides:

1. **Transitive Traversal:** Navigating recursive `DEPENDENCY_OF` relationships.
2. **Zero-Day Resilience:** Instant impact analysis for new CVEs.
3. **Code-to-Cloud Visibility:** Linking vulnerable code directly to the **ComputeInstances** where it is running.

Core Project Architecture

1. Data Ingestion (`loader.ipynb`)

- Loads transitive dependency data into the VPEM graph.
- Simulates multi-tier software lineage.

2. SBOM Analysis (`sbom.ipynb`)

- Identifies vulnerabilities in deep dependencies.
- Maps risk to internet-facing production infrastructure.

neo4j © Neo4j 2024

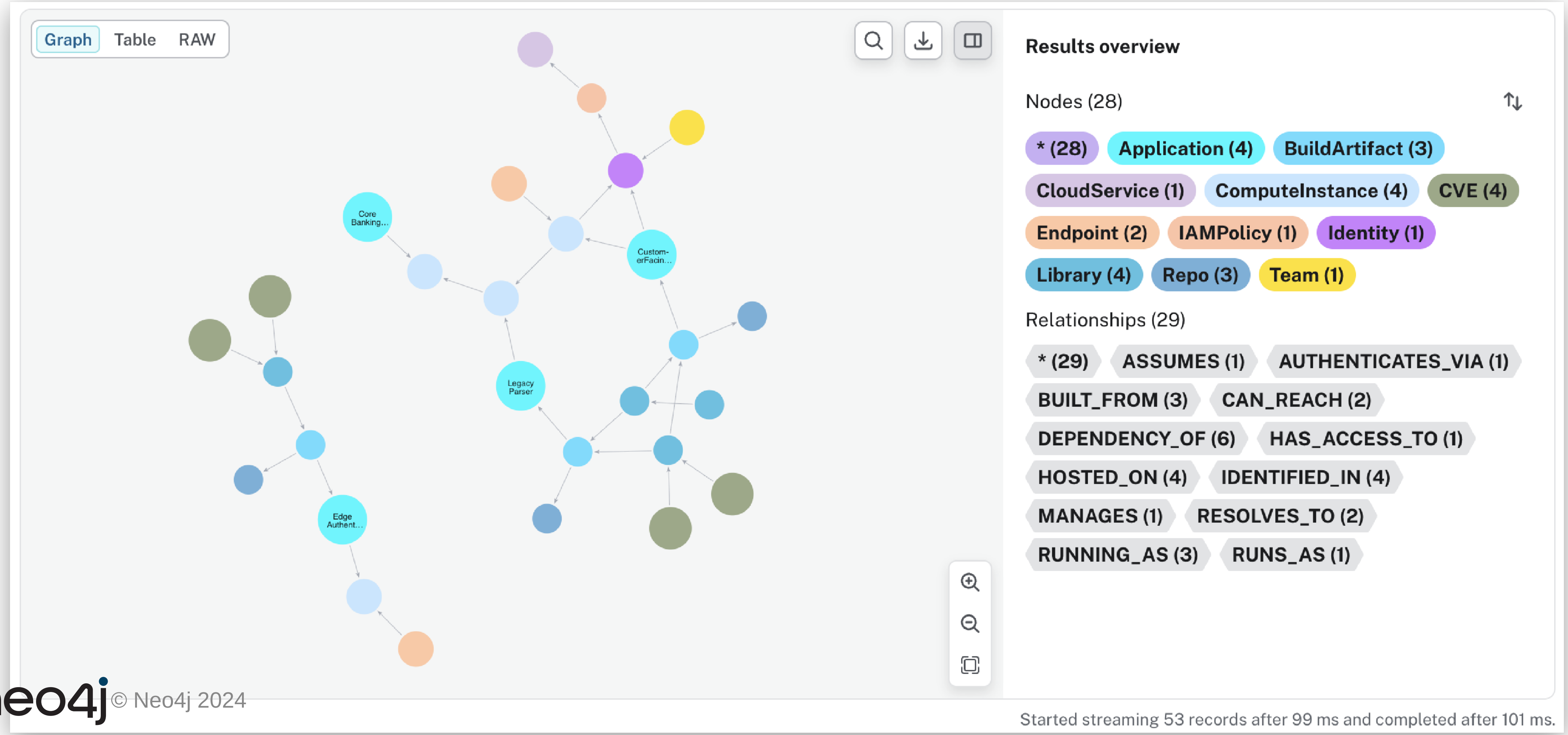
Analysing the Blast Radius

The Application View Graph

- **Traceability:** Map the path from a poisoned 3rd-party library to a business app.
- **Exposure:** Correlate vulnerable code with infrastructure data.
- **Real-time:** See which internet-facing servers are running "at-risk" artifacts.

© Neo4j 2024

The Blast Radius Graph



Results overview

Nodes (28)

* (28) Application (4) BuildArtifact (3) CloudService (1) ComputeInstance (4) CVE (4) Endpoint (2) IAMPolicy (1) Identity (1) Library (4) Repo (3) Team (1)

Relationships (29)

* (29) ASSUMES (1) AUTHENTICATES_VIA (1) BUILT_FROM (3) CAN_REACH (2) DEPENDENCY_OF (6) HAS_ACCESS_TO (1) HOSTED_ON (4) IDENTIFIED_IN (4) MANAGES (1) RESOLVES_TO (2) RUNNING_AS (3) RUNS_AS (1)

Started streaming 53 records after 99 ms and completed after 101 ms.

The Resulting Graph Schema

The model uses a Security Knowledge Graph to map three distinct layers:

- **Layer 1: Threat Intel** (CVE, NVD, CISA KEV).
- **Layer 2: Software Lineage** (Recursive `DEPENDENCY_OF` relationships).
- **Layer 3: Infrastructure** (Build artifacts $\longrightarrow$ Applications $\longrightarrow$ ComputeInstances).

neo4j © Neo4j 2024

The Graph Advantage: Impact Analysis

Unlike flat lists, Neo4j allows for Transitive Dependency Traversal:

- **Impact Analysis:** Instantly find every server affected by a new vulnerability.
- **Chokepoint Discovery:** Find internal shared components that, if patched, resolve the highest number of reachable risks.
- **Remediation Efficiency:** Prioritize by "reachability" and "exposure."

© Neo4j 2024

Key SBOM Queries

Transitive Risk Discovery (CVE → App)

Identify affected apps and their dependency chains for any CVE.

```
MATCH (v:CVE)-[:IDENTIFIED_IN]->(lib:Library)
MATCH path = (lib)-[:DEPENDENCY_OF*1..3]->(art:BuildArtifact)-[:RUNNING_AS]->(app:Application)
RETURN v.id AS cve,
       lib.name AS vulnerable_lib,
       app.name AS affected_app,
       nodes(path) AS dependency_chain
```

© Neo4j 2024

Visualise Log4Shell Chain

Directed Graph of Dependency Paths

Example for CVE-2021-44228 (Log4Shell) from library to applications.

```
MATCH (v:CVE {id: 'CVE-2021-44228'})-[:IDENTIFIED_IN]->(lib:Library)
MATCH path = (lib)-[:DEPENDENCY_OF*1..3]->(art:BuildArtifact)-[:RUNNING_AS]->(app:Application)
RETURN path
```

Render paths with `networkx` + `matplotlib` to show lineage and relationship types.

© Neo4j 2024

Reachability to Internet-Facing Hosts

Full Code-to-Cloud Trace

Trace vulnerable libs to running servers with public IPs for patch urgency.

```
MATCH (v:CVE)-[:IDENTIFIED_IN]->(l:Library)
MATCH path = (l)-[:DEPENDENCY_OF*1..3]->(:BuildArtifact)-[:RUNNING_AS]->(app:Application)-[:HOSTED_ON]->(ins:ComputeInstance)
WHERE ins.public_ip IS NOT NULL
RETURN v.id AS cve,
       app.name AS app,
       ins.name AS server,
       ins.public_ip AS ip
```

Outward System Integration

The insights from this analysis flow into:

- **DevSecOps (CI/CD):** Automatically fail builds that introduce critical transitive risks.
- **Security Operations (SOAR):** Prioritize patching for internet-facing hosts.

© Neo4j 2024

Questions?

GitHub: pedroleitao-neo4j/cyber-sbom

Reference: VPEM (Vulnerability Prioritization and Exposure Management)

neo4j © Neo4j 2024