ChatGPT

Using Threat Modeling and Semantic Graphs to Secure the Digital Supply Chain

White Paper co-authored by Dinis Cruz and ChatGPT Deep Research – May 2025

Introduction

Modern organizations rely on complex digital supply chains comprising not only their own systems, but countless third-party components and service providers. Each link in this chain – from open-source libraries to cloud vendors – introduces potential vulnerabilities that can ripple outward. This white paper presents a methodology to **validate and enhance supply chain security** by combining classical threat modeling techniques with **semantic graph representations**. Co-authored by Dinis Cruz (a pioneer in semantic knowledge graphs for cybersecurity) and ChatGPT Deep Research, it is aimed at the technical threat modeling community seeking scalable ways to manage multi-tier supply chain risks. We will explore how threat models can be extended recursively to suppliers and components, and how linking these models via knowledge graphs uncovers interdependencies, shared weaknesses, and attack propagation paths. In addition, we provide guidance on implementing this approach – referencing concepts from Cruz's *Project SupplyShield* – and illustrate how graph analytics (e.g. "blast radius" analysis) enable real-time validation and dynamic risk assessment across the supply chain.

The Complexity of Modern Digital Supply Chains

Today's digital supply chains are extraordinarily complex and deep. Organizations depend on myriad third-party software, open-source packages, APIs, and cloud services – which themselves depend on other fourth- or fifth-party components. Studies show that open-source software now composes 70–90% of the code in a typical application [1]. In practice, this means even a "simple" web app may rely on hundreds of indirect libraries and services maintained outside the organization. Each dependency is a potential source of compromise or failure. Indeed, companies face mounting risk from **fourth parties** (vendors' vendors): nearly 80% of businesses admit they urgently need to improve assessment of these downstream suppliers [2]. High-profile incidents have underscored the *systemic vulnerabilities* lurking in dependency webs. For example, the critical Log4j logging library ("Log4Shell" vulnerability) was embedded in countless products – over **100 million instances** of software were affected globally when this flaw was revealed [3]. Log4j's ubiquity (used by "millions of computers worldwide" [4]) meant a single weakness in one open-source component cascaded into a **global security crisis**.

An illustration of dependency risk: modern digital infrastructure often hinges on a single obscure component maintained externally [5].

Compounding the challenge, organizations often lack visibility beyond their direct third-party contracts. A vendor might outsource tasks to subcontractors or incorporate external SDKs, creating opaque *nested dependencies*. Traditional risk management struggles to map these relationships. Security teams are left asking: **Where are we exposed?** Which upstream components could introduce vulnerabilities into our environment? And how would a breach in one supplier propagate to us? Addressing these questions requires rethinking threat modeling to encompass the entire supply chain ecosystem, not just one system at a time.

Threat Modeling Across the Supply Chain

Threat modeling is a cornerstone of security design – typically used to identify and mitigate threats to a particular application or system. In a supply chain context, however, we must extend this practice **beyond the boundaries of a single organization or software module**. The key insight is that every supplier, component, or dependency in the chain can be treated as an entity with its *own* threat model. In other words, we model threats **recursively at each node** of the supply chain and then aggregate those insights to understand systemic risk.

In practical terms, this means when modeling a system's threats, one must consider not only the system's internal design, but also threats arising from external dependencies. For example, if your web application relies on a third-party payment service, your threat model should include the risk of that service being compromised or failing (and how an attacker might leverage that to harm your system). Likewise, if that payment service depends on a cloud infrastructure provider, the cloud provider's outages or breaches become part of *your* threat landscape. By viewing suppliers and sub-suppliers as integral parts of the attack surface, we create a **hierarchy of threat models**: the organization sits at the top, underpinned by models of each critical vendor, which in turn include models of their own dependencies, and so on. Any weakness in a lower tier can cascade upward. Industry experts emphasize that an enterprise's security is only as strong as the weakest link in its vendor ecosystem [6] – therefore our threat modeling must shine a light into those links.

To operationalize this, organizations can define threat scenarios at multiple levels: e.g. *"What if Supplier X's credential database is breached?"* or *"What if open-source library Y has a zero-day vulnerability?"*. These scenarios might be informed by frameworks like MITRE's supply chain attack patterns or past incidents (such as malicious package injections). Each scenario is essentially a mini threat model for that component, identifying entry points, potential adversaries, and controls. By cataloguing such threat models for key third/fourth-party elements, we build a knowledge base of risks that spans the entire supply network. The challenge, then, is how to manage and make sense of this sprawling collection of threat data. Manually keeping track of all the interconnections (who uses what, which vulnerability affects which component, which supplier has access to which system) quickly becomes infeasible with traditional spreadsheets or static diagrams. This is where **semantic graph representations** become invaluable.

Semantic Graphs: Linking Threat Models Across Dependencies

A **semantic knowledge graph** provides a powerful way to represent and navigate the complex web of relationships in a digital supply chain. In a graph model, each entity in the supply chain (an organization, a software application, a microservice, a third-party vendor, an open-source library, etc.) is a node, and each relationship ("depends on", "provides service to", "contains vulnerability", "interfaces with") is an edge. What makes the graph *semantic* is that these nodes and edges carry meaning defined by an ontology – a formal schema of the domain. In our case, the ontology might define entity types like *Supplier*, *Software Component*, *Data Asset*, *Vulnerability*, *Threat Scenario*, and relationship types like *uses*, *hosts*, *connects-to*, *vulnerable-to*, or *mitigated-by*. By encoding threat models into this graph (as nodes and links representing threats, mitigations, and impacts), we effectively create a **living map of the supply chain's risk landscape**.

Crucially, graphs can capture **multi-level dependencies and interdependencies** in a way that is difficult for linear documents. An illustrative example is given by an NSFOCUS study on software supply chain security: *"user A has App I installed, App I references open-source component A and B. If a remote code execution vulnerability exists in component A, the affected user A can be identified [via the graph]*... The

edges between nodes represent the dependency relationships … applicable to multi-level dependency analysis in software supply chain security." [7] In this graph, a path can be traced from a low-level component to a higher-level application and ultimately to an end-user or organization, reflecting how a single flaw propagates upward. The graph model naturally mirrors the *"long reference chain"* of modern software supply chains [8]. Nodes like an open-source library might be linked to dozens of different applications (representing *shared components*); if a vulnerability is found in that library, all those applications' nodes become connected to the vulnerability node. For instance, NSFOCUS notes that if *"open source component B is referenced by multiple apps… once component B is exposed with vulnerabilities, the influence scope will be expanded"*, and thus component B is marked as a high-risk node [9]. This is the kind of insight a semantic graph makes accessible: one glance can show that **a single point of failure** (like component B) spans many systems. Traditional threat models often consider one system at a time, but by linking them via a graph, we see the **global picture** – e.g., that a certain threat scenario appears in multiple places, or that two ostensibly separate vendors actually rely on the same SaaS sub-provider (an unseen common dependency).

From a structural perspective, this approach creates a *"graph of graphs"*. Each individual system or supplier's threat model can be seen as its own subgraph (with nodes for that system's assets, threats, controls, etc.). The supply chain knowledge graph links these subgraphs through dependency relationships. Dinis Cruz has described this concept as needing *"graphs of graphs, ontologies of ontologies, taxonomies of taxonomies"* – essentially multiple linked knowledge structures that evolve together [10]. By connecting threat models across nodes, we form a **holistic meta-model** of the entire ecosystem. This layered ontology allows zooming in and out: from the high-level supply chain map down to the granular details of a particular component's threat model. The semantic aspect ensures that relationships are meaningful and comparable across different parts of the graph. For example, a *"data breach"* threat scenario in a SaaS vendor's model might be linked to a *"data confidentiality"* impact node in your organization's model, aligning terminology so that automated reasoning can occur.

Equally important is the graph's ability to **adapt and grow**. Supply chains are not static – new dependencies emerge, vendors change, vulnerabilities get discovered weekly. Semantic graphs are well-suited to evolve with such changes. Unlike rigid tabular databases, graphs readily accommodate new nodes and relationships without a full redesign. As Cruz notes, graphs with an adaptable schema allow us to **refactor and extend** our knowledge representation over time [11]. The ontology can evolve – new node types (say, *Compliance Requirement* or *Threat Actor*) can be added as needed, and edges can flexibly capture new relationship types. This agility is crucial in the face of ever-shifting supply chain compositions and emerging threats. In summary, a semantic graph-based approach provides both the **structural clarity** and the **flexibility** needed to model complex, nested supply chain threats. It enables security teams to traverse the supply network, query "what connects to what", and discover non-obvious risk linkages (for instance, if two critical services rely on the same small cloud provider, a graph query would reveal that single point of failure). By establishing this connected view, organizations gain **situational awareness** that was previously elusive [10] – a map of risk that mirrors reality's complexity.

Implementation Guidance: Architectural Approach *(Project SupplyShield)*

Implementing threat modeling with semantic graphs requires a blend of process and technology. In this section, we outline a practical methodology and architecture, drawing on Dinis Cruz's previous work – especially his *Project SupplyShield* initiative – which provides a blueprint for using knowledge graphs to tackle supply chain security. The goal is to create an integrated platform (one might think of it as a

"Supply Chain Security Brain") that continuously ingests data, maps out risks, and supports decision-making. Key components of such an architecture include:

- **Inventory and Data Ingestion:** First, gather detailed data on all relevant supply chain components. This means compiling your **Software Bill of Materials (SBOM)** for internal applications (the list of all open-source libraries, modules, and their versions), as well as an inventory of third-party services, SaaS vendors, cloud infrastructure in use, hardware suppliers, etc. Importantly, also ingest meta-data about these components: vulnerability feeds (e.g. CVEs), security ratings of vendors, compliance attestations, known threat intelligence (for example, if a certain supplier was breached in the past or is associated with a particular threat actor group's interest). Automation is crucial here – leveraging tools to continuously pull data from sources like dependency scanners, vendor risk questionnaires, and threat intel databases will keep the graph up-to-date.

- **Semantic Knowledge Graph Store:** At the heart of the architecture is a graph database or knowledge graph platform where the above data is modeled. Establish a well-defined **ontology** for the supply chain security domain. For example, define node classes such as *Organization*, *Application*, *Service*, *Library*, *Vulnerability*, *Threat*, *Control*, *ComplianceRequirement*, etc. Define relationship types: e.g. *depends_on*, *uses*, *provides*, *vulnerable_to (exploited_by)*, *mitigated_by*, *located_in* (for geo-location of data centers), *owned_by* (for linking a service to its provider), and so on. Each node can have properties (e.g. a Vulnerability node might have properties like CVSS score, CWE type, date disclosed). By structuring the graph in this semantic way, we enable powerful queries later and ensure that as new data comes in, it fits into a cohesive structure. Notably, Cruz's approach to semantic knowledge graphs emphasizes that the **ontology is the operational core** – rather than a fixed schema that limits us, it should be an evolving framework that captures the organization's collective knowledge [12] [13]. In practice, this means the team should be prepared to iterate on the graph design, adding new node types or relationships as new use cases emerge (for example, incorporating *"LLM prompt"* nodes if analyzing AI supply chain risks, etc.).

- **Threat Model Integration:** How do we embed threat models into this graph? One method is to represent each threat scenario as a subgraph pattern. For instance, consider a threat model for a web application that identifies "SQL Injection on Customer Data DB" as a threat. In the graph, we might have a node representing that threat scenario, linked to the *Application* node (indicating the target), linked to a *Data Asset* node for the database, and linked to a *Vulnerability* node if one is known (e.g., a specific SQL injection CVE in an ORM library). Likewise, if a supplier has a threat model describing "Insider threat at Vendor X", that could be a Threat node linked to the *Vendor* node and to the *Impact* nodes that would result (data leak, service outage, etc.). In essence, each threat model element (threat, actor, asset, control, impact) becomes part of the graph, tied to the relevant supply chain entities. This might sound complex, but with a well-defined schema it becomes systematic. We might designate special relationships like *threatens* or *at_risk_asset* to connect threat nodes to asset nodes. The result is that the **graph now contains not just structural info (who depends on whom) but also analytical info (who threatens what, what controls exist)**. Weaving threat modeling outputs into the knowledge graph ensures that when we later traverse the graph, we can see not just connectivity but also context like "what could go wrong here" and "what security measures are in place here."

- **Decision Trees and Analytics Layer:** Data by itself isn't useful without analysis. Cruz highlights the use of **decision trees** and rules, often powered by GenAI and expert input, to interpret the graph and drive decisions [14]. In practice, one could implement a rules engine or graph analytics layer on top of the knowledge graph. For example, a simple decision tree might be: "If a

critical vulnerability (CVSS > 9) is found in a library that is used by a critical application and that application has no available patch, then flag as high risk and recommend action." These rules can be encoded in code or even represented as additional nodes (for complex multi-step logic). The combination of **human cyber-risk analysts and AI** can be leveraged here: AI (including large language models) can assist in generating and even reasoning over these rules. For instance, a GenAI could be prompted with the graph data to suggest new correlations ("Library Z is used in payment processing and has a high-severity flaw – likely a critical business risk"). The key is that by having all data in a structured graph, even an AI system can trace *"why is this supplier rated high risk?"* by following the graph edges to evidence [15]. This creates transparency and auditability that pure black-box AI systems lack. In Cruz's vision, **Project SupplyShield** leverages GenAI as an augmentation tool on top of the knowledge graph – to help maintain it and to generate insights from it [16]. For example, natural language queries could be posed to an LLM: "List all suppliers who lack multi-factor authentication and could impact customer data," and the LLM would traverse the graph (via an intermediate query) to answer with traceable references. Such capabilities turn the static risk register into an interactive, intelligent system.

- **User Interface and Visualization:** A practical implementation should include intuitive visualization of the supply chain graph and threat models. Security architects and risk managers should be able to see diagrams of how components link to each other. Graph visualization tools can show, for example, a selected supplier node and all its upstream and downstream connections. This is useful for **attack path analysis** – e.g., seeing that an attacker compromising Supplier X could then access API Y which touches your crown jewels database. Visual "maps" of the supply chain with overlay of risk scores or red flags can greatly aid comprehension for stakeholders (including non-technical executives). Dinis Cruz often refers to evolving graphs into *"maps that enable situational awareness"* [10] – a reminder that how we present the data is as important as the data itself. Consider including heatmap-style highlights on the graph (nodes colored by risk level) or filters to view only certain tiers or threat types.

- **Integration and Automation:** Finally, integrate this system with workflows and monitoring. The knowledge graph can feed alerts into a SIEM or GRC tool whenever a risk threshold is crossed (for instance, if a new 0-day vulnerability appears in a widely used library, automatically create an incident or task). Conversely, it can receive input from continuous monitoring: e.g., if a code repository scan finds a new dependency, a script updates the graph with that new node and links. Ideally, the system supports **real-time updates**, so that as soon as something changes in the supply chain (a new supplier onboarded, a configuration changed, a penetration test finding, etc.), the graph and threat models reflect it. The ultimate vision is a living, breathing supply chain risk model that is always current. Cruz suggests that using LLMs and automation is what finally makes it feasible to maintain these complex graphs with **repeatability, explainability, and provenance** [17] – tasks that would overwhelm purely manual efforts. By codifying knowledge and letting AI assist, the graph becomes a source of truth that can be queried and trusted for decision support.

In summary, the implementation involves building a **semantic knowledge graph platform** that federates all supply chain risk data, embedding threat modeling constructs into that graph, and using analytics (rules engines, AI, queries) on top to perform assessments. Project SupplyShield, for example, embodies this by combining a domain-specific knowledge graph with decision logic to perform supply chain risk management and compliance oversight [14]. A crucial practice is to move away from siloed spreadsheet-based vendor risk assessments to **dynamic, interconnected graphs** [16]. This shift brings immense advantages in scalability and insight: instead of static checklists, you get an ever-updating map where each piece of data strengthens the overall understanding of risk. Furthermore, every conclusion the system provides (e.g., a supplier's risk rating) can be traced to concrete evidence in the

graph – *"Supplier → lacked control X → violates compliance Y → leads to risk Z"* [15] – giving confidence to auditors and engineers alike that the assessment is grounded in data. By implementing such an architecture, organizations lay the groundwork for proactive and systematic supply chain security management.

Mapping the Blast Radius of Supply Chain Attacks

One of the most powerful capabilities enabled by a graph-based approach is **blast radius analysis** for any given component compromise. "Blast radius" in this context refers to the scope of impact or the reach of damage that would occur if a particular node in the supply chain were compromised or failed. In a complex vendor/product ecosystem, understanding blast radius is vital: it answers questions like *"If Library Q has a critical vulnerability, which applications (and thus which business services) are affected?"* or *"If Vendor X gets breached, how far could the attacker pivot into our systems?"*. With a traditional list-based inventory, answering these can take days of manual tracing. With a knowledge graph, it becomes a matter of running a graph traversal query.

For example, suppose a popular NPM package used in your software (Node1) is reported to be malicious. In the graph, you find the node representing that package. You can then traverse "upwards" (following *used_by* or *depends_on* edges) to see all software components that include it, then further up to see which applications or services those components are part of, and finally which business processes or products rely on those services. This chain of connections delineates the **blast radius** of the malicious package. Security teams can immediately identify *every affected system and stakeholder*. Google's open-source GUAC project (Graph for Understanding Artifact Composition) exemplifies this use case: it is designed to *"determine the blast radius of a bad package or a vulnerability"* and even suggest a remediation plan [18] . In other words, graph analytics can automatically map out all paths from the point of failure to any downstream dependents. Another real-world example is cloud security posture: companies like Wiz use a "security graph" to perform root-cause and blast radius analysis in cloud environments, automatically showing which resources would be impacted if one node is compromised [19] .

In our supply chain threat modeling graph, performing a blast radius analysis might involve algorithms like breadth-first search or depth-first search from the node of interest, or more sophisticated shortest path algorithms if we're looking at how an attacker might move. We can enrich this analysis with edge weights or filters – e.g., perhaps we weight edges by trust level (an API integration might be a low-privilege connection vs. a direct VPN access which is high-privilege). Then the "blast radius" is not just which nodes are connected, but also an assessment of how easily an issue propagates along those connections. For instance, if a third-party's credential is compromised, the blast radius includes all systems where those credentials are accepted; if a code library has a vulnerability, the blast radius includes all applications that import that library (and possibly their users' data). The knowledge graph already encodes these relationships, so the analysis is a matter of querying the graph with the right parameters.

Consider a concrete scenario: **Compromised Certificate Authority (CA)**. Say one of your software vendors relies on certificates issued by a particular CA, and that CA suffers a breach (its root certificate is no longer trustworthy). To find the blast radius, you'd query the graph for all services or software that trust that CA's certificates. The graph would reveal which applications embed that CA in their trust store, which users or systems depend on those applications, and so forth. This could quickly highlight, for example, that your Vendor A's product (which your company uses for email) will no longer securely communicate because the CA they use is untrusted – meaning your email communication integrity is at

risk. Without a graph, it might be unclear that Vendor A was using that CA in the first place; with the graph, this relationship is explicit.

Another scenario: **Zero-Day vulnerability in a common library** (similar to Log4j). Using graph analytics, one can instantly list all nodes of type *Application* or *Service* that have an edge to the vulnerable *Library* node. This was a huge challenge during Log4Shell – many organizations struggled to enumerate where they had Log4j. A well-maintained supply chain graph would have had a *Log4j* node with edges to all systems (internal or third-party) that include it, enabling a one-query answer to "where are we using Log4j?". In fact, some industry efforts like the OpenSSF's tooling and GUAC aimed to address exactly this by aggregating SBOM data into a graph [20]. As noted, understanding *"how one piece of software affects another"* is key to fully grasping your security posture [20]. Our approach formalizes that understanding.

It's worth noting that blast radius isn't only about technical components; it can also model **business impact propagation**. For instance, you could traverse from a compromised component to see which *business capabilities* or *missions* might be affected. If the graph links applications to business processes (e.g., *Payment Processing Service* → *supports* → *Revenue Collection*), then the blast radius of a technical failure can be expressed in business terms ("this incident could halt our ability to collect revenue from customers using payment method X"). This is extremely useful for communicating risk to executives in terms they understand.

In summary, using semantic graphs for threat modeling enables precise and rapid blast radius analysis. The moment a new threat is identified anywhere in the chain, you can **light up all connected nodes** to visualize the spread. This supports not only faster incident response (by immediately focusing on affected areas), but also proactive design: by studying these graphs, you might identify overly central dependencies (nodes with very large blast radii) and decide to add redundancies or stricter controls around them. As an example from the NSFOCUS research, they recommend identifying *"high-risk open source software nodes"* by looking at how widely used a component is and how frequently it's updated [21]. A component heavily used but infrequently updated is a ticking time bomb – the graph analytics can flag it so you can mitigate (e.g., find alternatives or push the community to patch). Thus, blast radius mapping is not just a reactive tool, but a strategic one to bolster resilience. Projects like SupplyShield envisage such capabilities at their core, ensuring that any potential compromise triggers an immediate graph traversal to assess impact and drive response planning.

Real-Time Validation and Dynamic Risk Assessment

A significant advantage of combining threat modeling with semantic graphs is the potential for **real-time validation and dynamic risk assessment** of the supply chain. In traditional third-party risk management, assessments are periodic (perhaps annual questionnaires or audits) and largely static. They might not account for rapid changes – a new vulnerability, a change in a supplier's security posture, or a newly discovered dependency in your code. In contrast, a graph-based model can be continuously updated and evaluated, functioning as a "living" risk register.

Real-Time Validation: Because the knowledge graph integrates data feeds, it can validate the security status of the supply chain in near real-time. For example, as soon as a new CVE vulnerability is published, it can cross-check if that CVE's affected product (say, OpenSSL library version X) appears in your dependency graph. If yes, the relevant nodes are flagged instantly. This dynamic lookup is far more efficient than waiting for the next scan or supplier self-report. Similarly, if a vendor publishes new compliance reports or security test results, those can update the graph (for instance, a *Control* node for "MFA enabled" on that vendor might flip from *false* to *true*). The graph thus serves as a continuously

validated model of reality – discrepancies or changes are immediately reflected. With this capability, we approach **real-time risk scoring**: at any given moment, the organization can query "what is my current aggregate supply chain risk?" and get an answer based on the latest data, not last quarter's information.

Dynamic validation also means we can simulate or anticipate changes. For instance, if considering onboarding a new supplier, you could insert a hypothetical node for that supplier with known attributes, and see how it connects in the graph and what new risks it might introduce (like linking to a region with high regulatory risk, or adding a dependency on an uncommon technology). This is essentially running *what-if analyses* on your supply chain model. If the simulation shows that adding Supplier Y would create a concentration of risk (e.g., you'd then have three critical services all relying on the same fourth-party hosting provider), you might reconsider or require mitigations. The semantic graph, with its rich encoding of relationships, supports these scenario analyses elegantly. It's akin to having a flight simulator for your supply chain – you can test turbulence (incidents) and see how the system handles it.

Dynamic Risk Assessment and Scoring: Traditional risk assessment often uses questionnaires and qualitative scoring. In our approach, we can derive **quantitative risk metrics** by leveraging graph analytics. For example, one could compute a "centrality" measure for nodes in the dependency graph – nodes with high centrality (many connections or critical bridging positions) might get a higher inherent risk score because they represent single points of failure. Likewise, one can propagate risk values through the graph: if a leaf node (like a low-level component) is rated high risk and it feeds into a parent node, the parent's risk can be incremented accordingly. This is essentially applying algorithms on the graph to calculate aggregate risk. An interesting avenue is using **graph neural networks or machine learning** on the graph to predict which suppliers are most likely to have an incident, based on patterns (though that's an advanced application beyond our scope here). At the very least, dynamic queries allow slicing the risk: "Show me all suppliers with risk score > 8 that directly connect to our crown jewels." This helps focus mitigations on the most impactful areas.

Real-time risk assessment also benefits from **human-in-the-loop AI**. As Cruz points out, the advent of GenAI means we can maintain and leverage these knowledge graphs at scales previously impossible [22]. Large language models can be used to parse news feeds or cyber threat intel feeds and automatically update the graph. Imagine an LLM reading a cybersecurity news article about a supply chain attack on a popular software provider – it could identify from context that "Company Z" (mentioned in the article) is one of your suppliers (matching against the graph data) and then flag that in the graph as an event node. This would alert your team to validate if that incident affects you. Furthermore, GenAI can help bridge terminology differences; not every vendor will describe things the same way, but an AI can normalize and map concepts to your ontology on the fly. Cruz's research discusses creating *"bridges between team ontologies without forcing standardisation"* [23] – which is highly relevant when integrating data from many sources in the supply chain. One team's "vendor tier 2" might be another's "fourth party"; an AI can reconcile that in the knowledge graph context so nothing slips through cracks.

Another dynamic aspect is **feedback loops**. Every incident or near-miss in the supply chain can be fed back into the model to improve it. For example, if a minor third-party outage occurred and wasn't initially flagged as high risk, one might update the threat model for that third party and increase its risk weighting. Over time, the system "learns" and becomes better at foreseeing issues. This approach aligns with Cruz's emphasis on *self-improving knowledge structures* [24]. Instead of a static set of controls, the threat models and graph adjust based on experience, much like how hazard maps are updated after each earthquake. We can also incorporate testing into validation: performing tabletop exercises or red-team drills against the supply chain model. For instance, simulate an attack on a supplier and see if the

graph (and the people/process around it) catch the scenario and have mitigations. This kind of dynamic validation ensures that the model isn't just theoretical but truly reflective of defense readiness.

Finally, dynamic assessment enables **continuous compliance**. Many regulations now (like those around critical infrastructure supply chain security, or new guidelines like NIST SP 800-161 for supply chain risk management) expect organizations to have up-to-date knowledge of their supply chain risks. A semantic graph can serve as evidence that you maintain ongoing diligence. Auditors could even be given query access to ask, "Have you identified all software in your environment affected by vulnerability X?" – and you could confidently demonstrate that with a query result, rather than manually assembling spreadsheets. The transparency of the graph approach means every risk decision is backed by data [25], which goes a long way in satisfying compliance and governance requirements.

In conclusion, real-time and dynamic aspects turn supply chain threat modeling from a static snapshot into a **continuous process**. The semantic graph acts as the connective tissue enabling this, while AI and automation act as the muscle that moves it in real-time. By adopting this approach, organizations can move towards a state of **active defense** in supply chain security – where risks are identified and addressed not just periodically, but as they emerge, and where the organization's risk posture is recalibrated on a continuous basis.

Conclusion

Securing modern digital supply chains requires a paradigm shift in how we approach threat modeling and risk management. The complexity – with its nested dependencies, third- to Nth-party relationships, and ever-evolving threat landscape – has outgrown traditional spreadsheets and one-off assessments. In this white paper, we have presented a methodology that combines **threat modeling at every layer** of the supply chain with the integrative power of **semantic knowledge graphs**. This approach allows us to map out the entirety of our supply chain in a living graph of nodes (systems, components, vendors, threats, controls) and edges (dependencies, interactions, vulnerabilities). By doing so, we make the invisible visible: interdependencies are illuminated, shared vulnerabilities are pinpointed, and potential attack propagation paths are charted.

We described how threat models can be recursively applied so that each supplier or component is not a black box but a knowable quantity in our risk analysis. By linking these models together, we create a holistic view where the impact of a single node's compromise can be understood in context of the whole. The use of semantic graphs ensures that this view is not a tangle of raw data, but an organized, queryable representation of knowledge. In essence, we advocate moving from static risk registers to a **knowledge graph of supply chain risk** – one that can be analyzed, queried, and updated in real time. As Cruz's research and Project SupplyShield emphasize, the ontology-driven knowledge graph approach is key to scaling this understanding, providing a framework that can evolve with the problem space [14] [10]. Notably, knowledge graphs also embed *explainability*; every link in the chain of reasoning for a risk decision can be traced in the graph, fostering trust and clarity [15].

The benefits of this methodology are both immediate and long-term. In the immediate term, organizations gain greatly enhanced **visibility**. It becomes possible to answer previously daunting questions (like "which of our critical services would be impacted if *this* vendor had an incident?") almost instantly. This directly improves incident response and prioritization of mitigations. We no longer fight supply chain fires in the dark – the graph lights up the network of connections so responders know exactly where to put resources. In the longer term, this approach enables **proactive risk management**. Patterns and hotspots can be discerned (e.g., a particular open-source library that constitutes a systemic risk, or a particular fourth-party that many of your vendors rely on). With that knowledge, you

can drive strategic initiatives – maybe sponsor improvements in that open-source project, or push vendors to have alternate providers. The approach also supports **collaboration and knowledge sharing**. An organization could share parts of its threat model graph (in a sanitized way) with key suppliers to jointly improve security, effectively linking graphs across organizational boundaries. In the future, industries might maintain collective supply chain knowledge graphs for common libraries or services, improving everyone's visibility. This aligns with the community efforts we see at OWASP, OpenSSF, and others in pooling together threat models and dependency data for the common good [26] [27].

Of course, adopting this methodology requires investment – in tools, in skills (graph thinking and threat modeling expertise), and in data collection. There will be challenges, such as keeping the data current and ensuring data quality across many sources. However, the convergence of technologies we have now – **graph databases, automation, and AI** – makes it more feasible than ever. As Dinis Cruz has noted, GenAI can act as a force multiplier, helping maintain and even populate these knowledge graphs in ways that were impractical just a few years ago [28]. We recommend starting with a pilot on a subset of your supply chain: perhaps focus on one critical product and its direct vendors, build a small graph and threat model, and use that to demonstrate value (e.g., you might uncover a previously unknown fourth-party risk through this process, which can galvanize support).

In closing, we stand at a juncture where supply chain security has become one of the defining challenges in cybersecurity. By applying rigorous threat modeling *recursively* and leveraging *semantic graph technology*, we can turn this challenge into a solvable problem. This approach provides a **methodology to validate** (by continuously checking assumptions against data) and **enhance** (by revealing gaps and prompting mitigations) supply chain security on an ongoing basis. Early adopters in this space – whether through internal projects or community initiatives – are already seeing the benefits of graph-based risk analysis [29]. As a community, if we embrace these techniques, we move closer to a world where even the most complex digital supply chains can be confidently understood and secured. The authors encourage fellow threat modeling practitioners to experiment with semantic graphs, share their learnings, and collaboratively develop the ontologies and open data that will strengthen all of our supply chains. Together, we can build the maps that illuminate our digital supply lines, and by understanding them deeply, ensure they remain resilient against the threats of today and tomorrow.

References: This paper has drawn upon and cited numerous sources to reinforce its concepts – including Dinis Cruz's LinkedIn articles and research on semantic knowledge graphs [10] [14], industry analyses on supply chain risk [2] [6], and technical studies like the NSFOCUS white paper on supply chain knowledge graphs [7]. These and other references are embedded inline throughout the text for the reader's further exploration. We highly recommend reviewing those materials (see the citation links) for deeper insights and related research, as they provide valuable context and validation for the methodology presented here. In particular, Cruz's *Project SupplyShield* and related writings offer a forward-looking perspective on how AI and graphs together can scale third-party risk management in ways that were not previously possible [17] [30]. Such works form the foundational ideas that inspired this white paper's approach. By marrying those innovative concepts with established threat modeling practices, we hope this paper serves as a comprehensive guide for practitioners aiming to bring their supply chain security to the next level. The path to secure supply chains is undeniably challenging, but with semantic threat modeling graphs as outlined, it is a path we can navigate with confidence and clarity.

---

1 WEF sounds alarm on software supply chain vulnerabilities, flags ...
https://industrialcyber.co/supply-chain-security/wef-sounds-alarm-on-software-supply-chain-vulnerabilities-flags-risks-in-open-source-and-third-party-dependencies/

2 'Fourth-party risk' rising during supply chain disruption | CFO Dive
https://www.cfodive.com/news/fourth-party-risk-rising-during-supply-chain-disruption/620590/

3 Log4j: The Aftermath and Lessons Learned - Critical Start
https://www.criticalstart.com/log4j-the-aftermath-and-lessons-learned/

4 What the Log4j vulnerability is, who is affected - NCSC.GOV.UK
https://www.ncsc.gov.uk/information/log4j-vulnerability-what-everyone-needs-to-know

5 The Rising Threat of Software Supply Chain Attacks: Managing Dependencies of Open Source Projects – Open Source Security Foundation
https://openssf.org/blog/2023/08/18/the-rising-threat-of-software-supply-chain-attacks-managing-dependencies-of-open-source-projects/

6 Unseen Dependencies - A Meta-Analysis of Fourth-Party Risk in the AI Supply Chain
https://www.linkedin.com/pulse/unseen-dependencies-meta-analysis-fourth-party-risk-dennis-henry-pms8e

7 8 9 21 29 Security Knowledge Graph | Drawing Knowledge Graph of Software Supply Chain and Strengthening Risk Analysis - NSFOCUS, Inc., a global network and cyber security leader, protects enterprises and carriers from advanced cyber attacks.
https://nsfocusglobal.com/security-knowledge-graph-drawing-knowledge-graph-of-software-supply-chain-and-strengthening-risk-analysis/

10 22 23 24 28 Beyond Static Ontologies: How GenAI Powers Self-Improving Knowledge Graphs
https://www.linkedin.com/pulse/beyond-static-ontologies-how-genai-powers-knowledge-graphs-dinis-cruz-ombpe?trk=public_post

11 12 13 Semantic Knowledge Graphs is the best definition for me so far :) For... | Dinis Cruz
https://www.linkedin.com/posts/diniscruz_semantic-knowledge-graphs-is-the-best-definition-activity-7282169143942860800-TgjN

14 17 GenAI-Driven Supply Chain Risk Management and Compliance | Dinis Cruz
https://www.linkedin.com/feed/update/urn:li:activity:7296674764733992961/

15 16 25 30 Gen AI and Knowledge Graphs to Scale Third-Party Risk | Dinis Cruz
https://www.linkedin.com/posts/diniscruz_gen-ai-and-knowledge-graphs-to-scale-third-party-activity-7296697059745714177-eBtN

18 20 27 guac
https://guac.sh/

19 Wiz Security Graph offers root cause analysis for cloud IR | Wiz Blog
https://www.wiz.io/blog/wiz-security-graph-enhances-cloud-incident-response

26 Threat Modeling the Supply Chain for Software Consumers – Open Source Security Foundation
https://openssf.org/blog/2023/09/27/threat-modeling-the-supply-chain-for-software-consumers/