ChatGPT

Graphs of Graphs of Graphs (G3) in Threat Modeling

Introduction

Modern cybersecurity architectures are highly complex, with myriad components, third-party services, and evolving threat landscapes. Traditional threat modeling approaches – often static diagrams or spreadsheets – struggle to capture this complexity over time. **Graphs of Graphs of Graphs (G3)** is an emerging concept that applies semantic knowledge modeling to threat modeling, enabling a more dynamic, interconnected representation of security knowledge. Instead of a one-off exercise, threat modeling becomes a continuous, data-rich practice that links multiple layers of information into an evolving "living" model [1]. By treating threat models as interlinked knowledge graphs, G3 provides a *single source of truth* for security risks and mitigations that is both machine-readable and human-explainable [1]. Co-authored by security researcher Dinis Cruz and ChatGPT Deep Research, this white paper defines G3 and illustrates how it transforms the way professionals structure and use threat models.

Understanding G3 and Semantic Knowledge Modeling

G3 Defined: *Graphs of Graphs of Graphs* refers to a paradigm where multiple knowledge graphs are connected to each other, creating an ecosystem of graphs across different layers or domains. In essence, each "graph" (e.g. a graph of an application's threats, a graph of attacker techniques, a graph of security controls) becomes a node or subgraph in a larger graph-of-graphs structure. This concept builds on semantic knowledge modeling principles: using **knowledge graphs** (nodes and relationships with contextual meaning) to encode information, and **ontologies** to formally define the types of entities and links. A G3 approach means we aren't confined to a single monolithic graph; instead, we maintain *modular, linked graphs* that together describe a complex security environment.

The theoretical foundation comes from the semantic web and knowledge graph research. By defining ontologies for the security domain, we can ensure that each graph "speaks the same language." For example, one ontology might define what a *Threat*, *Asset*, *Vulnerability*, *Control*, etc., mean and how they relate. G3 leverages the idea of **connected ontologies** – multiple domain-specific ontologies (for web apps, cloud, IoT, etc.) linked together – so that a threat model can integrate knowledge from many sources [2]. This means a threat model graph is not an island; it can reference industry taxonomies and standards. As Cruz's work on *Semantic OWASP* shows, using knowledge graphs allows us to customize and scale security knowledge across different contexts (for example, treating OWASP's Top Ten or other checklists as semantic data that can be linked in) [3]. In a G3 model, a node representing "SQL Injection" in one graph can link to a canonical definition in an industry ontology, ensuring consistency and shared understanding across systems.

Under the hood, G3 relies on **semantic interoperability** – ensuring that when different models use similar concepts, they are mapped to a common reference. A formal ontology provides this shared vocabulary. For instance, if one team's model labels a threat "SQL Injection" and another calls it "Database code injection," a common ontology would map both to the same concept [4]. Using standards like OWL/RDF from the semantic web, each element in a threat model graph can be linked to authoritative definitions (e.g. an OWASP or NIST entry), enabling automated reasoning and data exchange [5]. In summary, G3's theoretical basis is a fusion of knowledge graphs with well-defined semantics, allowing graphs of different scopes to connect into a cohesive whole.

G3 Applied to Threat Modeling: Multi-View, Multi-Graph Architecture

Applying G3 to threat modeling fundamentally changes how we structure and use threat models. Instead of a single static diagram, we maintain an interconnected set of graphs capturing multiple perspectives and layers of the security architecture. These graphs can represent different *views* of the system (business view vs. technical view, attacker perspective vs. defender controls, etc.) and different granularities (enterprise-wide threats vs. component-level threat models). Users can toggle between these views as layers of the overall graph [6]. For example, a CISO could view a high-level graph of business processes and associated threats, while an engineer zooms into a technical subgraph of a particular microservice's vulnerabilities [6]. Both views are derived from the same underlying interconnected data. This layered visualization is a core advantage of G3 – the ability to pivot the graph to see the aspect that matters to a given stakeholder, without losing traceability to other contexts.

Crucially, G3 enables linking what were previously siloed models. A threat model need not live in isolation for each system or vendor – graphs can be *linked across systems and organizations*. Consider a SaaS provider and its client: the provider's internal threat model might have a scenario "Data breach of SaaS app," which the client's threat model links to its own impact node "Loss of customer data confidentiality." In a G3 approach, these two graphs can be semantically connected so that both parties understand the shared risk in their own terms [7]. Similarly, if your application relies on a third-party API or open-source library, your threat graph can include nodes representing those dependencies. Those nodes might link to separate graphs (perhaps published threat models from the supplier, or community knowledge about that library's vulnerabilities). This creates a **graph-of-graphs**: your organization's threat graph, the vendor's threat graph, and an open vulnerability graph all linked. By structuring models this way, G3 supports a *supply chain view* of threats, where you can traverse connections to see how an issue in one component propagates to others.

At Photobox (a UK-based photo service company), Dinis Cruz demonstrated the power of graph-based threat modeling in practice. He refactored the company's risk workflows into a graph database, representing security data as queryable nodes and relationships [8]. This allowed the security team to visualize how risks, mitigations, and system components all interlink, and to query the "security graph" for patterns – for example, to find all systems that lacked a mitigation for a certain threat [8]. In essence, they built a graph-of-graphs where each system's issues were part of a larger network of risks. The result was not only better visualization of the security posture but also the ability to run graph analytics to spot gaps (e.g. identify an attack path from an external node to sensitive data that had no controls in between [9]). This shows how linking multiple sub-graphs (systems, threats, mitigations) provides holistic insight that a stand-alone threat model would miss.

Organic Evolution of Threat Graphs (File-Based Approach)

A key aspect of G3 is the **organic, evolving nature** of the graphs. Security models are not designed once and left on a shelf – they continuously grow and adapt as new threats emerge, architecture changes, and knowledge improves. G3 embraces this by making the graph structure easily editable, versionable, and extensible. In practice, this often means treating the graph data like code: storing it in a file system (e.g. as JSON or YAML files under version control) and updating it iteratively. Cruz advocates for managing threat models in structured text files checked into source control (Git), which provides a historical record of changes and integrates with development workflows [10]. Each change to the system – adding a feature, deploying a new service, addressing a vulnerability – can trigger an update to the threat graph. Through continuous integration (CI/CD) hooks, the graph is kept in sync with the live system, making threat modeling a living process rather than a one-time snapshot [11].

Storing the graphs as files (as opposed to only in a proprietary tool or database) has several advantages. It enables **serialization** of the graph (saving the whole knowledge graph or portions of it in a human-readable format), which makes it easier to review, share, and audit. It also enables using standard development tools to manage the model – for example, using diff/merge to see how the threat model changed between versions of the application. Dinis Cruz's experiments highlight that graphs can be visualized directly from these serialized forms (using graph visualization libraries or Graphviz), facilitating understanding and manual pruning of irrelevant nodes [6] [12]. If part of the model becomes obsolete (say a deprecated feature and its threats), that portion of the graph can simply be removed or archived in the file system, and the graph-of-graphs naturally heals – no rigid schema is "broken" by the change. In an **organically evolving graph**, new nodes and relationship types can be added as needed without a complete redesign of the model. This flexibility prevents ossification. In fact, an organically evolving knowledge graph "grows like a healthy ecosystem, continually adjusting and optimizing" and avoids the obsolescence of rigid top-down designs [13]. By embracing constant iteration, G3 ensures the threat model stays current and relevant, even as the underlying technology and threat environment change.

Another benefit of the file-based, modular approach is *programmability*. Developers can treat the threat model as data that their scripts and tools manipulate. For example, with the data in JSON, one can write a script to automatically flag any threat node that has no corresponding mitigation node and open a ticket in the issue tracker – effectively automating a security review step. In fact, this kind of integration is already feasible: if a threat node with no mitigation is found, an automated workflow can create a development ticket linking to that node [10]. Similarly, build pipelines can query the graph whenever new code is deployed, checking if any new component introduces unresolved high-risk threats [11]. G3 thus enables *Security as Code*: the graph model is part of the codebase, subject to CI/CD, and can trigger programmatic actions. This contrasts with monolithic, manual threat models that are hard to update – G3's organic model can be continuously refined with both human input and automated data feeds.

Linking Ontologies, Taxonomies, and Standards as Semantic Layers

A distinguishing feature of G3 is how it layers and links various security knowledge sources. In a G3 ecosystem, there isn't just one flat graph; there are interconnected semantic layers, each often corresponding to an ontology, taxonomy, or standard:

- **Threat Ontology Layer:** This layer defines what constitutes threats, vulnerabilities, attack techniques, etc. It can incorporate community taxonomies like STRIDE (for threat categories), MITRE ATT&CK (tactics and techniques), CWE (common weaknesses), and CAPEC (attack patterns) as structured knowledge [14]. For example, the concept of "Injection Attack" might be an ontology class with subtypes, linking OWASP Top Ten's SQL Injection entry and CWE-89 (SQL Injection weakness) as equivalents. In a G3 model, when an engineer adds a node for "SQL Injection" threat in their system graph, that node can link to this ontology layer, inheriting the standard definition and properties.

- **System Model and Asset Layer:** This layer represents the organization's architecture – e.g. classes for *Application*, *Service*, *Database*, *DataFlow*, etc. [14]. It captures how components are connected in the system. The threat graph nodes for assets in your model belong to this ontology. Because this layer is separate, it can be reused and extended (for instance, linking to a cloud provider's ontology for cloud resources). G3 allows the system model graph of one project to connect with another (for instance, linking a third-party service node in your graph to the third-party's own published model).

- **Mitigations and Controls Layer:** Here we capture security controls, countermeasures, and requirements (e.g. *Encryption*, *Input Validation*, *Firewall*, or standards like NIST 800-53 controls, OWASP ASVS requirements, etc.) [15]. Each control in the graph can link to industry standards. For instance, a *Multi-Factor Authentication* control node could link to a NIST guideline or ISO control category. By linking threat nodes to control nodes ("Threat X is mitigated by Control Y"), and control nodes to standard catalogs, organizations achieve traceability from threats to compliance requirements.

- **Risk & Impact Layer:** This can model business impact, likelihood, incident records, and other risk management concepts [16]. For example, *Impact* nodes (like "Data Breach Impact") link to both threat nodes (that could cause that impact) and asset nodes (that would be affected), and perhaps to regulatory compliance nodes (GDPR, etc., if that impact has regulatory significance). This layer helps tie technical findings to business consequences.

These layers in G3 are *interconnected*. A single threat instance in your environment graph will have semantic links upwards into the abstract layers (to know what kind of threat it is, what standard category it falls under) and sideways to other graphs (e.g. linking to the same threat in a supplier's model or a historical incident database). Because the ontologies are modular, we can plug new ones in or update them without disrupting the whole model [17]. For example, if a new industry taxonomy of AI threats emerges, you can add it as another graph layer and map its entries to your existing nodes (say linking "Prompt Injection" in the AI taxonomy to the appropriate threat nodes in your system). G3's layered design thus fosters **reuse** of knowledge: you define a concept once in its ontology graph, and reference it in countless threat models. This overcomes the limitation of monolithic models by avoiding duplicate, inconsistent definitions. Instead of every project maintaining a separate list of "threat types" or control libraries, they all draw from (and contribute to) the shared semantic layers.

Case Study: G3 in Action for Supply Chain Threat Modeling

To make the G3 approach concrete, consider a real-world scenario of **digital supply chain security** – securing an application that depends on numerous external components (open-source libraries, SaaS APIs, cloud services, etc.). A G3-based threat model for this scenario would include multiple interconnected graphs: the organization's internal system graph, graphs for each critical supplier or dependency, and global knowledge graphs of vulnerabilities.

For instance, imagine your company uses an open-source logging library (similar to the real-world Log4j example). In a traditional model, you might just list "third-party library risk" in a spreadsheet. In a G3 model, you would have a node representing that library in your system graph, which is linked to an external graph of known library vulnerabilities. When the infamous "Log4Shell" vulnerability was disclosed, that vulnerability node in the global knowledge graph would already be linked to the library node – instantly showing that your application is affected (because the graphs are connected). The impact was massive (affecting "over 100 million instances" of software globally) precisely because so many systems had that library [18]. With G3, the moment the global vulnerability graph is updated, your local threat model graph lights up via the connection, and you can trace the impact: "Library X is used in

Application Y; vulnerability Z in X thus threatens Y.” This automated traceability is invaluable in supply chain risk management.

Now consider a third-party SaaS service your product relies on (e.g. a payment processing API). Your G3 threat model will include a subgraph for “Payment Service” with assumptions and requirements (like “Payment service must enforce security control A, B, C”). Through collaboration or public disclosures, you might obtain a simplified threat model from that SaaS provider. G3 allows linking their model to yours. So if their model has a node “Service outage” or “Data breach in Payment Service,” you can connect that to nodes in your model (like “Payment service unavailable” threat, or an *Impact* node for potential data loss). This alignment means an automated reasoning engine can see the chain: *threat at supplier -> impact on our system*, and even check if mitigations exist on both sides. In our example, your model might show that a data breach at the supplier would compromise your customer data node – a relationship that should spur a mitigation like data encryption or contractual obligations for breach notification. By linking graphs, **personalized views** emerge: the supplier views that threat as loss of their data, you view it as compromise of your asset, but the G3 graph connects them so both parties can trace the full path.

An implementation of these ideas is reflected in **Project SupplyShield**, a prototype by Cruz that combines a domain-specific supply chain knowledge graph with automated decision logic [19]. SupplyShield’s graph maps relationships between organizations, software components, known vulnerabilities, threats, and compliance requirements, continuously updated by data feeds. Instead of relying on static questionnaires or spreadsheets for vendor risk, it uses graph analytics. For example, one can query: “show all suppliers lacking multi-factor authentication who can impact customer data.” Using the interconnected graph, such a query traverses through Supplier nodes, finds those with a relationship indicating no MFA control, and then checks which are connected to your critical data assets – returning an answer with a traceable path for each risk [20]. Analysts can visualize an attack path originating from a supplier: the graph might reveal that *Compromising Supplier X* could lead to unauthorized access of *API Y*, which connects to *Database Z (your crown jewels)* [21]. This level of **traceability** – seeing how an attack can propagate across interconnected systems – is a direct result of the G3 approach. It helps prioritize risks and responses in a complex supply chain. Furthermore, automation built on the graph can proactively flag changes: if a new vulnerability is announced in a library, or a supplier’s security rating drops, the system can alert security teams and even recommend mitigations by searching the knowledge graph for applicable controls.

The supply chain case study highlights how G3 delivers **personalization, traceability, and automation** in threat modeling. The model is *personalized* to the organization’s specific technology stack and suppliers, but it’s not built in isolation – it is enriched by linking to external sources (vuln databases, partner data) that provide relevant context. Every element in the graph carries provenance (you know exactly which supplier or component it came from, and even why it’s considered a risk, thanks to links to vulnerability entries or requirements), ensuring traceability. And many processes are automated: from ingesting data (scans, feeds) to producing outputs (alerts, graph-based reports), the G3 model reduces manual effort and catches issues that humans might overlook in a spreadsheet.

Benefits of G3: Personalization, Traceability, and Automation

G3 offers significant advantages over traditional threat modeling approaches by leveraging its modular, semantic graph architecture:

- **Personalization:** Security models can be tailored to the context of a particular organization, system, or even persona. Because G3 links global knowledge to local models, each team member views the threats most relevant to their domain. For example, developers can focus on a microservice's threat subgraph with coding-level details, while an executive sees an abstracted view highlighting business impacts [6]. The same underlying data serves multiple needs. This personalized perspective extends to using AI assistants: one can query the graph in natural language (e.g. "What are the top threats to our payments service?") and receive answers specific to their environment [22]. The graph's rich semantic context makes such customization possible.

- **Traceability:** Every node and edge in a G3 model carries context and history, which greatly aids traceability. It's possible to trace a threat from a technical origin up through business impacts and even up to compliance obligations in one connected view. Links between layers (threat → control → standard) mean you can start at a vulnerability and follow the chain to see which asset it affects, what controls mitigate it, and which policy or standard is satisfied by those controls. In the Photobox example, the team could trace missing mitigations across all systems by querying the graph [8]. In supply chain scenarios, traceability means you can pinpoint how a risk in a supplier's environment could cascade into yours [21]. This end-to-end visibility is nearly impossible with document-based models. Moreover, because threat models are stored in version control, you have an audit trail of changes: one can trace *when* a certain threat was identified or mitigated by examining the commit history [10].

- **Automation:** By making threat models machine-readable and modular, G3 unlocks extensive automation opportunities. Security checks that used to be manual can be encoded as graph queries or scripts. For instance, you can automatically scan the knowledge graph for policy violations or missing links (as noted earlier, finding any threat node without a mitigation and opening a Jira ticket is trivial with a graph query [10]). Integration with CI/CD means the graph is updated and checked on every code push or infrastructure change [11]. Automated reasoning engines can even infer new insights, like identifying an attack path or suggesting a control that was not explicitly documented, by traversing the graph relationships [9]. The result is a threat modeling process that scales with DevOps and agile development – always running in the background, providing continuous feedback. This level of automation and integration is a stark contrast to large monolithic threat models that often fall out-of-date; instead of being a quarterly exercise, threat modeling becomes a continuous security *program*.

Beyond these specific benefits, G3 as an approach inherently promotes **modularity, reuse, and programmability**. Each small graph (be it for a component, a knowledge domain, or a standard) can be maintained independently and then assembled as needed – encouraging reuse of threat knowledge across projects. One team's threat scenario can be reused by another team facing a similar technology, simply by linking to the same nodes in the shared graph database, rather than duplicating data. The use of open formats (JSON files, ontologies) and APIs allows engineers to integrate the threat model into other tools and write custom automations, treating the model as code. This modularity and machine-readability address the limitations of past approaches that were often too rigid or too reliant on specific tools.

Conclusion

Graphs of Graphs of Graphs (G3) represents a visionary step in threat modeling, moving the discipline from static and siloed documents to agile, interoperable knowledge ecosystems. By combining semantic ontologies with practical file-based workflows, G3 enables threat models that are *organic* (constantly evolving), *connected* (linking multiple perspectives and external knowledge), and *actionable* (driving automation and decision-making). This approach overcomes the scalability issues of monolithic models – instead of one giant graph that is cumbersome to maintain, we have a constellation of smaller graphs that together form a rich picture of security. The G3 philosophy echoes the way living knowledge bases work: continuously refined by both human experts and machines, adapting to new information without losing historical context [13].

For cybersecurity and threat modeling professionals, adopting G3 means embracing a more **transparent, modular, and collaborative** process. Threat models become not just internal documents, but shareable knowledge assets: for example, companies can publish portions of their threat graph as part of security transparency initiatives, confident that common ontologies will let others interpret them correctly [23] [5]. Teams can integrate their models, mapping where their security postures intersect – much like linking puzzle pieces – which is crucial in supply chain security and cross-organization risk management. Ultimately, G3 in threat modeling is about elevating security knowledge into a distributed graph that everyone can query, contribute to, and learn from. It turns threat modeling into a living, semantic graph of graphs – one that grows smarter and more effective with each linked node, each disclosure, and each automated query, driving us toward a more secure and transparent digital ecosystem.

Sources: This paper references insights from Dinis Cruz's recent work on semantic knowledge graphs and threat modeling [8] [2] [6] [21], demonstrating how G3 concepts are being applied to advance security practices in the real world.

---

[1] [6] [10] [11] [12] [14] [15] [16] [17] [22] Advancing Threat Modeling with Semantic Knowledge Graphs - Dinis Cruz - Documents and Research
https://docs.diniscruz.ai/2025/05/29/advancing-threat-modeling-with-semantic-knowledge-graphs.html

[2] [3] [4] [5] [8] [9] [23] Threat Models as Mandatory Disclosures: A Vision for Security Transparency - Dinis Cruz - Documents and Research
https://docs.diniscruz.ai/2025/05/29/threat-models-as-mandatory-disclosures__a-vision-for-security-transparency.html

[7] [18] [19] [20] [21] Using Threat Modeling and Semantic Graphs to Secure the Digital Supply Chain - Dinis Cruz - Documents and Research
https://docs.diniscruz.ai/2025/05/30/using-threat-modeling-and-semantic-graphs-to-secure-the-digital-supply-chain.html

[13] From Top-Down to Organic Evolving Graphs, Ontologies, and Taxonomies - Dinis Cruz - Documents and Research
https://docs.diniscruz.ai/2025/03/29/from-top-down-to-organic-evolving-graphs-ontologies-and-taxonomies.html