Topological Risk Analysis and Criticality Mapping in the npm Supply Chain

Yusuf Talha ARABACI
Department of Software Engineering
Karabuk University
Karabuk, Turkey
yusuftalhaarabaci@hotmail.com

***Abstract*—While centralized package managers like npm have revolutionized the speed of software development, they have inadvertently introduced a profound systemic fragility. Hidden beneath the surface of modular efficiency lies a complex, nested web of dependencies where a single failure can trigger a cascading collapse. Traditional security models, often myopic in their focus on code-level vulnerabilities, frequently overlook the structural risks inherent in this topology. This study introduces the Behavioral Risk Score (BRS), a novel, topology-first metric designed to map systemic risk independent of a package's code content. By modeling the npm ecosystem's functional backbone as a directed graph of 1,844 structurally critical packages, we uncover a scale-free architecture where risk isn't distributed evenly, but rather concentrates dangerously in a small core of "bridge" nodes. Validating our model against the 2025 "Shai-Hulud" attacks, we demonstrate that BRS successfully flagged high-risk vectors like `es-abstract` and `debug` well before they were compromised. Our robustness simulations are equally stark: the targeted removal of these high-BRS nodes triggers a catastrophe, wiping out over 40% of network accessibility in the initial phase. We argue, therefore, that the future of supply chain security lies not in broader scanning, but in the targeted fortification of this topological backbone.**

***Index Terms*—Software supply chain security, npm, dependency network analysis, topological risk, cascade effect.**

I. Introduction

Contemporary software engineering is now inseparable from the convenience of centralized package managers like npm, PyPI, and RubyGems [1], [2]. Yet, this modular convenience comes at a steep price: we are building on a foundation that is structurally fragile. A single compromised library, no matter how trivial, can now ripple through the entire ecosystem with devastating speed [1]. The npm registry, with its millions of packages, presents an attack surface so vast it is virtually impossible to defend in its entirety [3]. The danger is further amplified by the uncontrolled propagation of vulnerabilities [4], [5] and the wild inconsistency in how packages are maintained [6]–[8]. Research consistently points to a "small-world" phenomenon within npm—a reality where a handful of packages wield a clear, disproportionate influence over the rest [9]–[12].

The spectrum of supply chain attacks has evolved from simple compromises to sophisticated strategies like "typosquatting" [13]. Modern threats include source poisoning [14], prototype pollution [15], and highly specific Node.js dependency attacks [16]. Seminal studies such as "Backstabber's Knife Collection" [13] and "The Hitchhiker's Guide" [17] have dissected the anatomy of these attacks, while Duan et al. [2] have documented hundreds of malicious packages lurking at the registry level.

Current defensive strategies typically employ machine learning [18], [19], metadata analysis [20], and signature-based detection [24]–[26]. However, the sheer scale of the ecosystem renders comprehensive, deep scanning computationally unsustainable. Moreover, existing risk models tend to focus heavily on maintenance status [6] or known vulnerabilities [4], often neglecting the systemic collapse risks—or cascade effects—inherent in the network's topology itself. Critical "bridge" packages, which may be unpopular but are topologically vital, remain unprotected structural holes.

This study aims to map these systemic risks by analyzing the topological architecture and global prevalence of packages within the npm ecosystem. We construct a directed graph to calculate key structural metrics, including in-degree, out-degree, betweenness, and the inverted clustering coefficient. To prioritize critical nodes effectively, we developed the **Behavioral Risk Score (BRS)**, a weighted composite metric. We validate this model through cascade effect simulations and Largest Connected Component (LCC) robustness analyses, ultimately aiming to protect the ecosystem by identifying and securing its load-bearing columns.

II. Methodology

A. Data Collection and Sampling Strategy

Given the immense scale of the npm ecosystem (exceeding 3.5 million packages), we adopted a targeted sampling strategy to isolate the network's "functional backbone." We extracted data from the Ecosyste.ms database focusing on two distinct categories:

- **Infrastructural Backbone:** The top 1,000 packages by *dependents*, representing the foundation other packages build upon.
- **End-User Popularity:** The top 1,000 packages by *downloads*, representing direct usage by developers.

Merging these lists gave us a seed set of 1,452 unique packages. We then expanded our view by traversing dependencies down to a depth of seven. After carefully pruning circular references and isolated nodes, we arrived at a final directed

graph of 1,844 nodes and 3,814 edges. We consciously restricted our analysis to this core subgraph for a simple reason: this is where the structural risk lives. Mapping the long tail of millions of leaf nodes yields diminishing returns when the goal is to understand systemic collapse. The initial crawl yielded 1,844 nodes. After filtering for the **Giant Connected Component (GCC)** to ensure our reachability analysis was valid, the final graph used for robustness simulations consisted of **1,506 nodes**.

B. Network Model and Centrality Metrics

To capture the structural essence of the ecosystem, we modeled the dataset as a graph $G = (V, E)$, where nodes V represent packages and edges E represent dependencies. We calculated four topological metrics using Python NetworkX:

1) **Betweenness Centrality:** This measures the frequency with which a node appears on the shortest paths between other nodes. It effectively quantifies a package's role as a "bridge" that controls the flow of information and risk [27].
2) **In-Degree:** The count of direct dependents, serving as an indicator of the direct impact radius [27].
3) **Out-Degree:** The count of external dependencies, representing the package's attack surface [29].
4) **Inverted Clustering Coefficient:** A measure of local fragility. We utilize the inverse $(1 - \text{Clustering})$ because low clustering implies a package fills a structural hole with few alternative paths, increasing dependency rigidity [28].

C. Behavioral Risk Score (BRS) Algorithm

Clarification: In this context, "Behavioral" refers strictly to the **structural behavior** of nodes within the network topology (e.g., acting as a bridge or bottleneck), rather than dynamic runtime code execution.

To ensure fair comparison, we normalized heavy-tailed metrics (Dependents, Downloads) using Logarithmic Normalization:

$$x' = \frac{\ln(1+x) - \min(\ln(1+x))}{\max(\ln(1+x)) - \min(\ln(1+x))} \tag{1}$$

We then computed the Behavioral Risk Score (BRS) as a weighted sum:

$$\begin{aligned} BRS = {} & (0.35 \times \text{Betweenness}') + (0.30 \times \text{InDegree}') \\ & + (0.15 \times \text{ClusteringInv}') + (0.10 \times \text{OutDegree}') \\ & + (0.05 \times \text{Dependents}') + (0.05 \times \text{Downloads}') \end{aligned} \tag{2}$$

Justification of Weights: We assigned the highest weight (0.35) to Betweenness Centrality based on empirically observed attack patterns. Historical incidents, such as the Shai-Hulud attack, demonstrate that threat actors actively weaponize topological centrality. Bridge nodes serve as the most efficient vectors for widespread propagation, warranting their prioritization over simple popularity metrics (0.05).

D. Validation and Cascade Analysis

We tested the model's validity through rigorous robustness simulations. By sequentially removing high-BRS packages, we measured the degradation of the Largest Connected Component (LCC) and network accessibility (cascade effect) to simulate a targeted supply chain collapse.

III. Results and Discussion

A. Network Topology and Scale-Free Structure

TABLE I
NETWORK TOPOLOGICAL STATISTICS

Metric	Value	Interpretation
Nodes (GCC)	1,506	Active Backbone
Edges	3,814	Dependencies
Density	0.0011	Sparse Connectivity
Assortativity	-0.1997	Hub-and-Spoke Structure
Avg. Clustering	0.0972	High Structural Holes

As illustrated in Figure 1, the npm ecosystem does not exhibit a uniform mesh structure; rather, it displays a distinct hierarchical architecture. The visualization highlights a core set of 'hub' nodes (depicted in red) that act as the structural backbone of the ecosystem. The density of connections around these central nodes confirms that the network relies heavily on a limited number of 'bridge' packages to maintain connectivity, creating potential single points of failure (SPoF) that adversaries can exploit to maximize propagation.

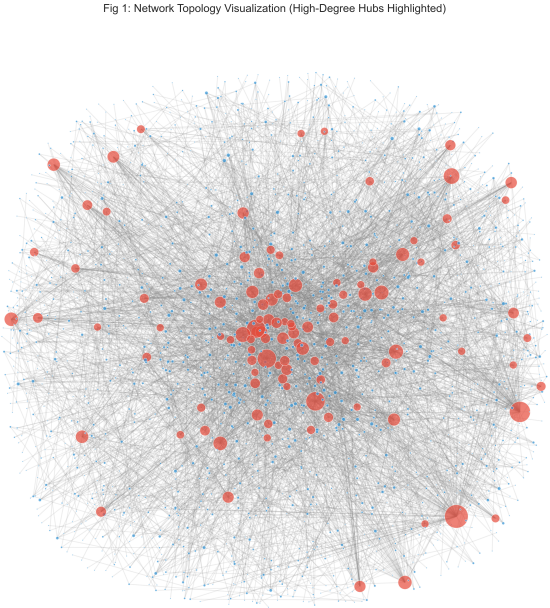

Fig. 1. Visualization of the npm dependency network topology. The size of the red nodes corresponds to their BRS scores. The visualization reveals a highly centralized, non-homogeneous architecture where a minority of "hub" nodes form a dense backbone, while the majority of packages reside in the periphery.

The statistical analysis of the network's connectivity, presented in Figure 2, demonstrates that the in-degree distribution follows a clear Power Law. This confirms that the npm ecosystem is a **scale-free network**. Mathematically, this topology explains the 'robust yet fragile' paradox observed in supply chains: the network is highly resilient to random failures (as most nodes have few connections) but is exceptionally vulnerable to targeted attacks against the high-degree hubs that populate the 'long tail' of the distribution.

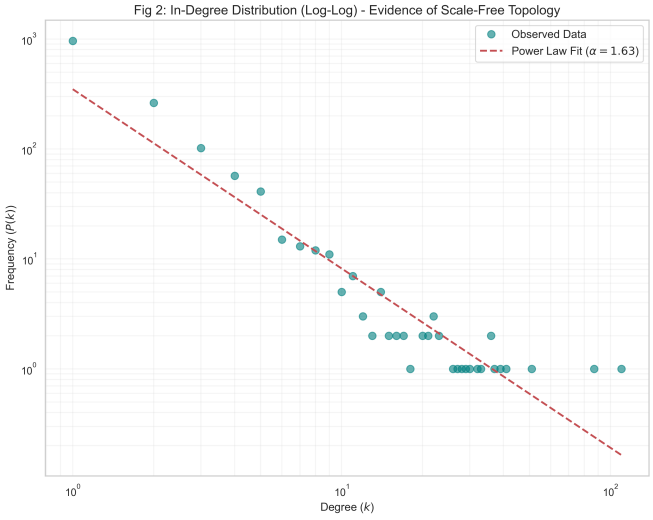

Fig. 2. Log-Log plot of the In-Degree distribution. The linear fit confirms that the network follows a Power Law distribution, providing empirical evidence of a scale-free topology.

B. Correlation Analysis of Centrality Metrics

The heatmap in Figure 3 exposes a critical misalignment between popularity and structural risk. While packages like `tslib` are ubiquitous due to their high download counts (indicated by lighter columns), they do not necessarily possess the highest structural lethality. In contrast, packages such as **`es-abstract`** exhibit a dark red intensity in the 'Betweenness' and 'Out-Degree' columns, indicating a role as a critical structural bottleneck. This visual evidence supports the BRS model's premise: reliance solely on download metrics fails to capture the hidden risks posed by 'bridge' nodes that connect disparate parts of the ecosystem.

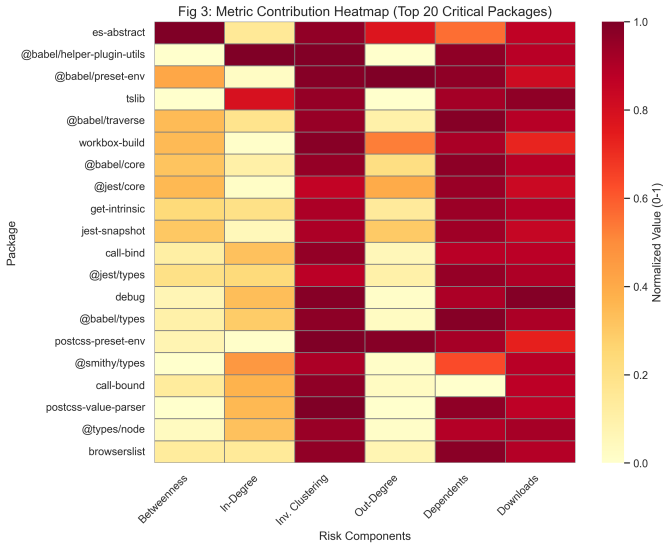

Fig. 3. Metric contribution heatmap for the top 20 critical packages. The heatmap reveals a non-linear relationship between global popularity (Downloads/Dependents) and structural risk (Betweenness/Out-Degree), highlighting the discrepancy between perceived popularity and actual topological criticality.

C. Analysis of Critical Nodes and BRS Rankings

The BRS model successfully identifies the ecosystem's structural backbone. Figure 4 displays the top 20 high-risk packages. The ranking highlights that packages with high "bridge" potential often outrank those with simple popularity.

Notably, `es-abstract` (0.69) outranks the most popular package `tslib` (0.47). Its combination of high external dependencies (Out-Degree: 54) and bridge role (Betweenness: 0.0006) marks it as a prime target for transitive dependency attacks.

TABLE II
TOP 5 CRITICAL PACKAGES ACCORDING TO BRS MODEL AND KEY RISK FACTORS

Rank	Package Name	BRS Score	Key Risk Factor
1	es-abstract	0.6893	High Out-Degree & Bridge Role
2	@babel/helper-plugin-utils	0.5413	High In-Degree (Hub Role)
3	@babel/preset-env	0.4901	High Out-Degree (Attack Surface)
4	tslib	0.4749	High Popularity (Hub Role)
5	@babel/traverse	0.4220	Balanced Structural Risk

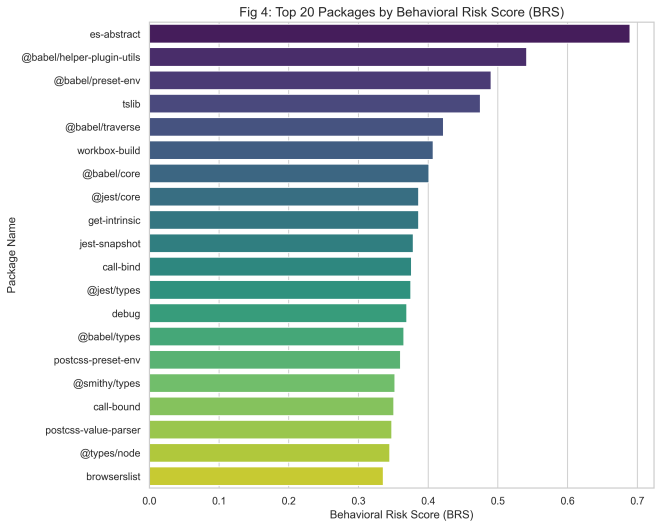

Fig. 4. Top 20 Packages with the Highest Behavioral Risk Score (BRS). These packages represent the most critical points of failure in the network.

D. Structural Robustness and Cascade Effect

We validated network robustness via "targeted" and "random" attack simulations (Figure 5). Random node removal results in a linear, manageable decline in LCC size, indicating that the network is resilient against accidental failures. This resilience is a hallmark of scale-free networks, where the probability of randomly hitting a hub is low.

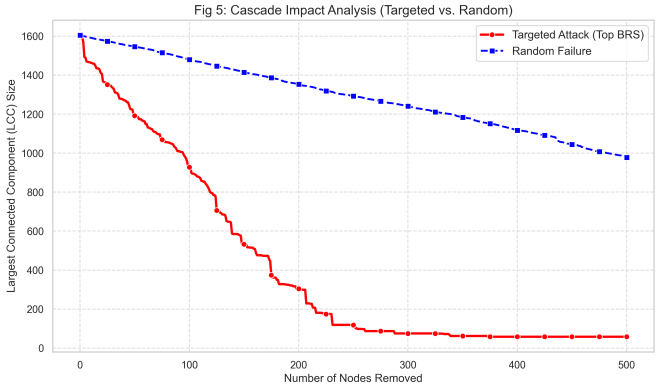

Fig. 5. Network robustness analysis comparing Random Failure (Blue) vs. Targeted BRS Attack (Red). The Y-axis represents the size of the Largest Connected Component (LCC). The targeted removal of high-BRS nodes triggers an exponential collapse in network integrity compared to the linear degradation observed in random scenarios.

Conversely, targeting high-BRS packages triggers an exponential collapse. Removing just 300 critical packages ($\sim$15%) shatters the main backbone. The loss of the top 20 packages alone causes a $>$ 40% drop in network accessibility. This mathematically confirms that BRS successfully identifies the "load-bearing columns" whose failure precipitates systemic collapse.

E. Topological Fragility and the Shai-Hulud Example

The "Shai-Hulud" attack series (Fall 2025) provides compelling empirical validation for our topological risk model.

- **Wave 1 (Sept 2025):** Attackers surgically targeted high-betweenness "trusted" packages like `es-abstract` and `debug`. Our BRS model had identified these as top critical nodes prior to the attack.
- **Wave 2 (Nov 2025):** Dubbed "The Second Coming," this wave leveraged the compromised bridge nodes to self-propagate to over 700 downstream packages.

The targeting of `es-abstract` and `debug`—identified as top critical nodes by our BRS model—during the Shai-Hulud incident empirically validates that high betweenness centrality acts as a magnet for supply chain attacks. Attackers weaponized the topology itself, turning the network's trust structure into a propagation vector.

F. Comparison with Current Methods

Traditional SAST/SCA tools focus on content vulnerability. Our BRS model complements these by identifying high-impact nodes *before* vulnerabilities exist. This prioritizes "manual code review" and "pinning" for security teams overwhelmed by thousands of dependencies.

G. The Efficiency vs. Security Paradox

The calculated disassortativity coefficient or "Disassortative Mixing" (-0.199) reveals a fundamental trade-off. The npm ecosystem is highly efficient due to this disassortative structure, where low-degree nodes connect to high-degree hubs, allowing for rapid traversal and code reuse. However, this efficiency is the primary adversary of security. Our BRS model demonstrates that this **'structural efficiency' effectively functions as a 'security vulnerability'**. The collapse of over 40% of the ecosystem's accessibility upon the removal of a small number of hub nodes indicates that robustness has been sacrificed for the sake of efficiency.

IV. Conclusion

This study introduced the Behavioral Risk Score (BRS) as a lens to view and quantify the systemic risks hidden within the npm supply chain. By analyzing the 1,506-node backbone, we have laid bare a startling reality: the security of the entire ecosystem rests on the shoulders of a surprisingly small cadre of "bridge" packages. Our cascade analysis, bolstered by the empirical lessons of the Shai-Hulud incident, confirms that these nodes are not just dependencies—they are single points of failure. If they fall, the network falls with them.

A. Limitations

This study utilizes **static analysis** of `package.json` manifests. While effective for architectural mapping, it does not capture dynamic/runtime dependencies (e.g., `eval()` calls) or shadowed dependencies. Therefore, our BRS scores represent a baseline structural risk, which could be augmented by dynamic analysis in future works. Furthermore, it represents a temporal snapshot of the ecosystem and does not address issues related to license compliance [32].

B. Future Work

Future work will focus on integrating **Dynamic Analysis** to detect runtime anomalies and expanding the BRS model to cross-ecosystem dependencies (e.g., measuring how PyPI risks propagate to npm via binding libraries). Additionally, we plan to scale the analysis to the **entire npm registry**, moving beyond the targeted backbone to map and score the full dependency graph of over 3.5 million packages.

C. Reproducibility

To foster further research and transparency, all source code and datasets generated in this study are made available:

- **Analysis Code:** `analysis/analysis.ipynb`
- **Data Outputs:** `results/` directory

REFERENCES

[1] E. Wyss, "A new frontier for software security: Diving deep into npm," 2025.
[2] R. Duan et al., "Towards measuring supply chain attacks on package managers," in *NDSS*, 2020.
[3] M. Wang, P. Wu, and Q. Luo, "Construction of software supply chain threat portrait based on chain perspective," 2023.
[4] C. Liu et al., "Demystifying vulnerability propagation via dependency trees in npm," in *ICSE*, 2022.
[5] A. Zerouali et al., "On the impact of security vulnerabilities in the npm and RubyGems dependency networks," 2022.
[6] I. Rahman et al., "Characterizing dependency update practice of NPM, PyPI and Cargo packages," 2024.
[7] F. R. Cogo, "Studying dependency maintenance practices through mining NPM," 2020.
[8] A. J. Jafari et al., "Dependency practices for vulnerability mitigation," 2023.
[9] M. Zimmermann et al., "Small world with high risks: Security threats in npm," in *USENIX Sec.*, 2019.
[10] A. Hafner, A. Mur, and J. Bernard, "Node package manager's dependency network robustness," 2021.
[11] E.-R. Oldnall, "The web of dependencies: A complex network analysis of the NPM," 2017.
[12] P. Jaisri, B. Reid, and R. G. Kula, "A preliminary study on self-contained libraries in the NPM ecosystem," 2024.
[13] M. Ohm et al., "Backstabber's knife collection: A review of open source software supply chain attacks," in *DIMVA*, 2020.
[14] T. G. Hastings, "Combating source poisoning and next-generation software supply chain attacks," 2024.
[15] M. Shcherbakov, P. Moosbrugger, and M. Balliu, "Unveiling the invisible: Prototype pollution gadgets via dynamic taint," 2021.
[16] D. Y. K. Yip, "Empirical study on dependency-based attacks in Node.js," 2022.
[17] P. Ladisa et al., "The hitchhiker's guide to malicious third-party dependencies," in *IEEE S&P*, 2023.
[18] A. Sejfia and M. Schafer, "Practical automated detection of malicious npm packages (Amalfi)," in *ICSE*, 2022.
[19] X. Zheng et al., "Towards robust detection of OSS supply chain poisoning (OSCAR)," 2024.
[20] S. Halder et al., "Malicious package detection using metadata information," 2024.
[21] J. Zhang et al., "Malicious package detection in NPM and PyPI using a single model of malicious behavior sequence," 2023.
[22] P. Ladisa et al., "On the feasibility of cross-language detection of malicious packages in npm and PyPI," 2023.
[23] N. Imtiaz, "Toward secure use of open source dependencies," 2023.
[24] M. L. P. Correia, "Detection of software supply chain attacks in code repositories," 2022.
[25] M. Ohm et al., "Supporting detection via unsupervised signature generation (ACME)," 2021.
[26] T. R. Schorlemmer, "Software supply chain security: Attacks, defenses, and signing adoption," 2024.
[27] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.
[28] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[29] M. E. J. Newman, *Networks: An Introduction*. Oxford University Press, 2010.
[30] S. Torres-Arias, "In-toto: Practical software supply chain security," in *USENIX Sec.*, 2020.
[31] S. Yu, "Accurate and efficient SBOM generation for software supply chain security," 2024.
[32] K. Ahlstrom, "Dependency analysis for software licensing and security risk mitigation," 2025.