Unveiling Dependency Clusters and Security Risks in the Maven Ecosystem Using Neo4j Analysis

George Lake
Department of Computer Science
Idaho State University
Pocatello, ID, USA
georgelake2@isu.edu

Minhaz Zibran
Department of Computer Science
Idaho State University
Pocatello, ID, USA
zibran@isu.edu

***Abstract*—We conduct a comprehensive cluster-based analysis of dependency risks in Maven Central by combining community detection and risk assessment techniques. By applying the Leiden algorithm to over 15 million artifacts, we identified approximately 67 thousand distinct clusters and developed a composite risk assessment framework that integrates CVE metrics, update patterns, and inter-cluster connectivity. Our analysis revealed that while most clusters exhibit low to moderate risk, the ecosystem's structure amplifies vulnerabilities through highly connected clusters acting as dependency hubs. These hub clusters, some containing over 1.5 million nodes, create critical pathways for vulnerability propagation despite moderate individual risk scores. We provide actionable recommendations for monitoring high-risk clusters and bridge nodes to enhance ecosystem stability. This analysis offers organizations a systematic approach for understanding and mitigating systemic dependency risks in large-scale software ecosystems.**

***Index Terms*—Maven Central, Dependency Analysis, Community Detection, Software Ecosystem, Vulnerability Analysis, Risk Assessment, Leiden Algorithm, Bridge Nodes, Principal Component Analysis**

I. Introduction

Software dependencies are fundamental to modern development, enabling code reuse and faster cycles. However, this interconnectedness also introduces risks, as vulnerabilities can propagate through complex dependency networks. Maven Central, one of the largest Java package ecosystems with over 15 million artifacts and 134 million dependencies, exemplifies these challenges.

Despite the importance of managing dependency-related risks, organizations often focus on individual package vulnerabilities, overlooking systemic risks arising from interconnections. This study addresses this gap by analyzing the Maven Central ecosystem through a cluster-based approach, uncovering how artifacts group naturally and the risks these groupings pose.

Using the **Maven Dependency Graph** [1] which models dependencies temporally, this research leverages the Goblin framework and a Neo4j graph database to explore three key questions:

RQ1: Can we deduce different clusters from Maven Central's dependency graph, and how do these clusters interact with one another?

RQ2: How can clustering be used to identify high-risk clusters in the Maven Central ecosystem?

RQ3: What potential risks and vulnerabilities can be identified by analyzing these high-risk clusters, and how can this analysis inform organizations about dependency-related risks in the Maven ecosystem?

By applying the Leiden algorithm, we identified 67,669 clusters with a modularity score of 0.704, indicating strong structural boundaries. A composite risk score, derived through Principal Component Analysis (PCA), highlights vulnerabilities within these clusters, offering actionable insights into dependency management. Our findings reveal that a small number of dominant clusters exert disproportionate influence, concentrating risks and creating systemic vulnerabilities.

This study contributes a novel methodology combining network analysis, risk assessment, and empirical validation, providing practitioners with recommendations for risk mitigation and researchers with a framework for analyzing package ecosystems.

The remainder of this paper is organized as follows: Section 2 presents related work and background. Section 3 details our methodology and data collection approach. Sections 4, 5, and 6 present our findings for each research question respectively. Section 7 discusses the implications and limitations, and Section 8 concludes with future research directions.

II. Methodology and Data Collection

Our study utilizes the Maven Central Dependency Graph created by the Goblin framework [1], which provides a comprehensive temporal model of dependencies enriched with metrics for ecosystem analysis. We specifically utilized the Goblin Maven dump from 08/30/24 with metrics, downloaded from Zenodo (link: https://zenodo.org/records/13734581)

A. Dataset

We worked with a Neo4j graph database containing the complete Maven Central ecosystem structure. The database includes detailed metrics for artifacts and releases as noted:

1) **Normalized CVE count:** The total number of CVEs in the cluster divided by the number of releases in the cluster, producing a per-release vulnerability metric.

2) **CVE Severity Score:** The number of CVE metrics added and a measure of the severity of vulnerabilities, rated on a scale from 1(LOW) to 4(CRITICAL), reflecting the impact of security issues.
3) **Freshness Score:** The average age of the artifacts, with higher scores indicating a higher risk due to outdated components.
4) **Speed:** Measures the average time between updates, highlighting maintenance responsiveness.
5) **Popularity Metrics:** Indicators of the level of adoption and use within the ecosystem, where higher popularity can imply greater impact if vulnerabilities are present.

B. Approach

We began by modeling the Maven Central Dependency Graph in Neo4j and created a projected copy optimized for analysis. Using the Neo4j Graph Data Science (GDS) library [2], we applied the Leiden community detection algorithm to identify clusters of artifacts, which formed the foundation of our analysis. We measured the modularity of these clusters to evaluate their structural cohesiveness and to understand the grouping of artifacts within the ecosystem. To support further analysis, we assigned a unique *communityID* property to each node, identifying the clusters.

We selected the Leiden algorithm for our community detection analysis due to its improved performance in both speed and quality of the partitions. Unlike the Louvain algorithm, which can result in communities that are arbitrarily poorly connected or even disconnected [3], [4], the Leiden algorithm guarantees well-connected communities and continues to refine partitions to ensure local optimality [4]. This characteristic is particularly crucial for our study, which relies on accurately identifying cohesive clusters.

To calculate the composite risk score for each cluster, we applied Principal Component Analysis (PCA) to derive empirical weights for the metrics. PCA was chosen for its ability to handle the inherent complexity and interdependence of various risk factors objectively. As noted by Greco et al. [5], PCA offers a robust framework for developing composite indices by systematically identifying the most significant sources of variance in a multidimensional dataset. This process assigned weights that reflect the natural importance of each metric, free from subjective bias, making it a reliable method for our risk assessment.

Finally, we analyzed the identified clusters to uncover key patterns and potential vulnerabilities. Our focus included examining bridge nodes that serve as critical connectors within the dependency graph and assessing how outdated or vulnerable artifacts could affect other components through dependency chains. This analysis provided actionable insights into risk propagation and highlighted areas within the ecosystem that require closer monitoring and intervention.

To support reproducibility, further research, and more data that could not fit in this paper, we provide our analysis code and documentation in a public repository **https://github.com/georgelake2/maven-central-cluster-analysis**

III. Analysis and Findings

A. Identifying Clusters and Their Interactions

The application of the Leiden algorithm partitioned the Maven Central Dependency Graph into 67,669 distinct clusters, achieving a modularity score of 0.704. This score indicates well-defined structural boundaries within the graph, demonstrating the effectiveness of the clustering approach in identifying cohesive artifact communities.

Cluster Characteristics: The majority of clusters are small, with fewer than 100 nodes, while a small number of dominant clusters, such as cluster 26959 (1.8 million nodes), exhibit significantly large sizes. This distribution highlights a skewed ecosystem where a few large clusters hold disproportionate influence. These dominant clusters often act as hubs, connecting numerous smaller clusters and serving critical roles in the ecosystem's stability. Fig. 1 visualizes the distribution of cluster sizes, underscoring this disparity.

Inter-Cluster Relationships: Analysis of inter-cluster connections revealed intricate dependency relationships, with some flusters acting as bridges between otherwise disconnected groups. For instance:

- Cluster 48568 has 1.79 million dependencies connecting it to other clusters, including cluster 29418, which itself serves as a hub with 1.44 million connections to cluster 48568. This is shown in Table I.
- Specific artifacts also serve as bridge nodes. For example artifact *org.apache.camel:apache-camel* links cluster 48568 to cluster 13965 via 311 unique connections.

Structural Insights: Dominant clusters display high edge density, suggesting that updates or disruptions within these groups could ripple through the ecosystem. Conversely, smaller, isolated clusters pose a smaller systemic risk due to their limited connections.

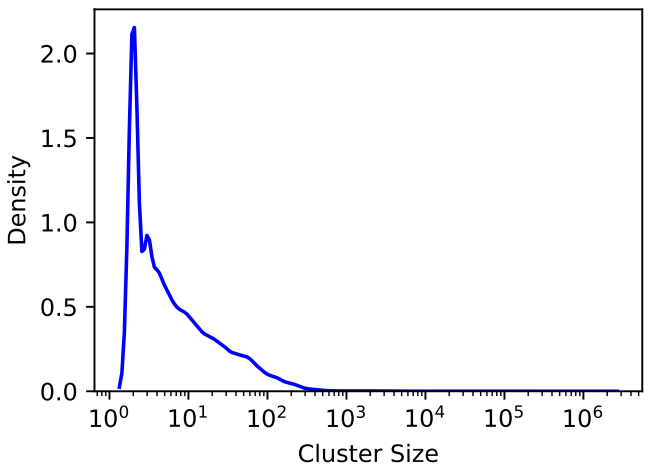

Fig. 1. Kernal Density Estimate of Cluster Sizes

Main Finding RQ1: Our analysis revealed distinct clusters in the Maven Central Dependency Graph, which exhibit numerous inter-cluster connections, both directly and through critical bridge nodes.

B. Assessing High-Risk Clusters Using a Composite Risk Score

The PCA results assigned the following weights:

- Normalized Total CVEs: 37.65%
- Average Severity Score: 38.35%
- Average Freshness Score (Days): 6.88%
- Average Popularity: 16.56%
- Average Speed (Days): 0.55%

Risk Distribution: The composite risk score revealed that most clusters showed low to moderate risk, with a few clusters that were identified as significantly high-risk. Fig 2 illustrates the distribution, highlighting a concentration of risk in specific clusters.

Key findings include, as shown in Table I:

- **Cluster 37666:** Highest risk score (1.00) due to elevated CVE counts and severity, but remains isolated with no external dependencies
- **Cluster 48568:** Moderate risk score (0.41) but critical to the ecosystem due to its size (1 million nodes) and interconnectivity (1.79 million dependencies).
- **Cluster 29418:** A central hub with a risk score of 0.56, reflecting it extensive connectivity to other clusters.

Insights into Risk Factors: CVE-related metrics (total count and severity score) were the most influential contributors to cluster risk. Small, isolated clusters (like 37666) may pose localized threats but are unlikely to impact the broader ecosystem. Large, interconnected clusters amplify systemic risk, making them high-priority candidates for monitoring and intervention.

Main Finding RQ2: A few disconnected clusters had high-risk scores, while the majority of clusters had low to moderate risk scores. Some of the moderate-risk clusters contain millions of nodes and are connected to multiple other clusters in the ecosystem.

C. Potential Risks and Vulnerabilities

Our analysis revealed three key categories of risk within the Maven ecosystem: security vulnerabilities, maintenance issues, and systemic risks from cluster interconnectivity.

Security Risks: Analysis of high-risk clusters revealed significant security threats, with some clusters showing critical vulnerability concentrations. Cluster 37666, despite its small size (179 nodes), exhibits the highest risk score (1.0) with an average CVE severity score of 2.49 and 15.15 normalized CVEs. Bridge nodes between clusters create critical pathways for vulnerability propagation. For instance, artifacts like *org.apache.camel:apache-camel* connect major clusters through hundreds of dependencies, creating potential security bottlenecks.

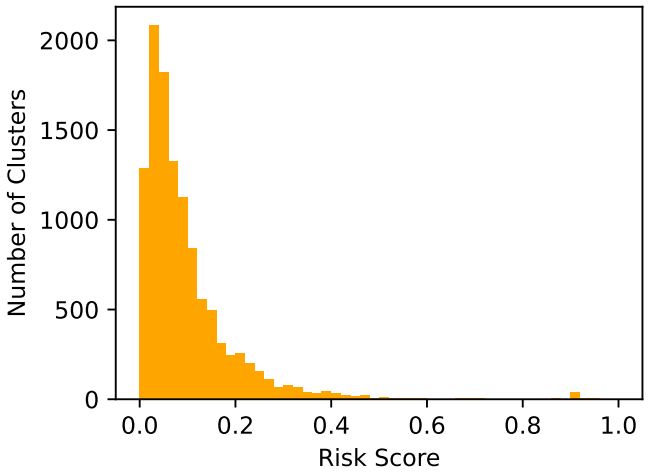

Fig. 2. Distribution of Risk Scores for Clusters

Maintenance and Technical Debt: High-risk clusters demonstrate concerning maintenance patterns. For example, cluster 43639 (risk score 0.80) shows extended intervals between updates, averaging 157 days. Key artifacts like *org.keycloak:keycloak-core* exhibit update delays of 6-12 months, significantly increasing vulnerability exposure. This aligns with findings from Imtiaz et al. [6], who found that delayed vulnerability remediation often stems from insufficient automated monitoring and inconsistent security tool usage.

Ecosystem Health Implications: Our analysis identified several systemic risks:

- Concentration of Risk: Large clusters like 29418 (1.29 million nodes) act as critical hubs with over 1.4 million inter-cluster dependencies.
- Update Cascades: Complex dependency chains in clusters like 48568 mean vulnerabilities can affect thousands of downstream projects.
- Bridge Node Vulnerabilities: Critical connecting artifacts create single points of failure that could impact multiple clusters.

Risk Mitigation Recommendations: Building on Imtiaz et al.'s [6] findings regarding the effectiveness of automated vulnerability detection, we recommend:

1) Implement automated dependency monitoring focused on high-risk clusters.
2) Prioritize updates for bridge nodes and highly connected artifacts.
3) Establish regular security audits for clusters with high risk scores.
4) Support critical open-source projects in high-risk clusters through community involvement.

Our findings emphasize that effective risk mitigation requires both automated analysis and ecosystem-wide collaboration. Organizations should focus on clusters with moderate to high risk scores and those serving as major dependency hubs, with the greatest potential for vulnerability propagation.

TABLE I
SELECTED CLUSTER METRICS - NOT ALL INCLUSIVE

Cluster ID	Risk	Size	CVE	Severity	Update	Freshness	Popularity	Average Speed	Target Cluster	Connections
37666	1.00	179	15.15	2.49	91	2,028	0	0.05	N/A	0
43639	0.80	143	11.23	2.94	157	712	0	0.07	N/A	0
28947	0.56	561,882	2.83	2.59	1,357	879	6.66	0.10	29418	241,801
29418	0.56	1,293,045	5.26	2.94	1,765	793	2.38	0.10	48568	1,447,774
67551	0.44	168,704	4.65	2.79	1,370	1,723	0.31	0.08	48568	41,254
48568	0.41	1,073,498	2.28	2.82	1,473	828	2.92	0.11	29418	1,791,786
28940	0.34	1,632,903	1.59	2.52	664	455	2.65	0.12	29418	293,896
14097	0.30	458,680	1.79	2.70	1,252	657	1.16	0.10	28947	299,194
26959	0.26	1,795,221	1.29	2.77	1,674	597	0.66	0.09	29418	502,251
30270	0.25	1,028,568	1.22	2.87	1,773	1,309	0.30	0.10	29418	451,654

Main Findings RQ3: Highly connected, large clusters and bridge nodes represent critical vulnerabilities in the Maven ecosystem, where their interconnectedness amplifies the impact of any vulnerability. This structural characteristic of massive, interlinked clusters suggests focused risk mitigation strategies to ensure ecosystem stability.

IV. THREATS TO VALIDITY

PCA Metric Weighting: While PCA provides a robust, data-driven method for determining metric weights, it introduces certain limitations that may impact the validity of the risk assessment model. PCA relies on the variance structure of the data to assign weights, which can underemphasize metrics that exhibit low variance even if they represent substantial risk. Additionally, PCA does not incorporate domain-specific knowledge or contextual factors, potentially overlooking critical nuances that an expert-informed model might adress.

Leiden Algorithm: The algorithm provides high-quality community detection and exhibits non-deterministic behavior. Similar to most community detection algorithms, it is possible to produce slightly different cluster assignments, particularly for nodes at community boundaries.

Data Completeness: Our analysis relies on the Maven Dependency Graph snapshot from August 30, 2024. While comprehensive, this snapshot may not capture the most recent changes in the ecosystem. Additionally, the graph may not include dependencies hosted outside Maven Central.

Metric Selection: The choice of metrics used in our risk assessment model, while based on established literature and practical considerations, may not capture all aspects of dependency risks.

V. RELATED WORK

Dependency Graph Analysis: Several studies have explored the structure and evolution of software dependency networks. Kikas et al. [7] analyzed the npm ecosystem's dependency network. Zimmerman et al. [8] demonstrated how highly popular packages can create systemic risks. Our work extends these findings to the Maven ecosystem, providing insights into how dependencies cluster and propagate risks.

Community Detection in Software Ecosystems: The application of community detection algorithms to software systems has gained attention in recent years. Subelj and Bajec [9] demonstrated the effectiveness of community detection for understanding software structure, while Hora et al. [10] applied similar techniques to study API usage patterns. Our study builds upon these approaches by applying the Leiden algorithm to the Maven ecosystem.

Risk Assessment in Package Ecosystems: Previous research has established various approaches to assessing risk in software ecosystems. Decan et al. [11] studied the impact of security vulnerabilities in npm packages, demonstrating how vulnerable packages affect ecosystem portions. Our research extends this by using PCA analysis to develop a composite risk score combining multiple metrics.

Empirical Studies on Ecosystem Health: Previous work has focused on measuring ecosystem health. Valiev et al. [12] studied npm package sustainability, while Kula et al. [13] investigated dependency update patterns. Our study contributes new metrics for assessing ecosystem health through cluster analysis and risk assessment.

VI. CONCLUSION

This study provides a comprehensive analysis of the Maven Central ecosystem, leveraging graph-based methods and risk assessment frameworks to address critical challenges in dependency management. By modeling the ecosystem as a Neo4j graph and applying community detection algorithms, we identified 67 thousand distinct clusters, revealing a skewed distribution where a small number of large clusters hold disproportionate influence. These dominant clusters, along with critical bridge nodes, form the backbone of the ecosystem but also represent significant points of vulnerability.

The composite risk assessment framework identified several vulnerabilities in the ecosystem. While some small, isolated clusters showed high individual risk scores, the greatest systemic risks emerge from large, well-connected clusters that combine moderate risk scores with extensive interconnectivity.

These findings have important implications for dependency management practices. We recommend actively tracking bridge nodes between major clusters, particularly those connecting clusters with high-risk scores or large node counts. Further, large clusters should be priority targets for security monitoring.

REFERENCES

[1] D. Jaime, https://orcid.org/0000-0002-7503-4606, View Profile, J. E. Haddad, https://orcid.org/0000-0002-2709-2430, View Profile, P. Poizat, https://orcid.org/0000-0001-7979-9510, and View Profile, "Goblin: A Framework for Enriching and Querying the Maven Central Dependency Graph," *Proceedings of the 21st International Conference on Mining Software Repositories*, Apr. 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3643991.3644879

[2] "The Neo4j Graph Data Science Library Manual v2.12 - Neo4j Graph Data Science." [Online]. Available: https://neo4j.com/docs/graph-data-science/2.12/

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, Oct. 2008. [Online]. Available: https://dx.doi.org/10.1088/1742-5468/2008/10/P10008

[4] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities." *Scientific Reports*, vol. 9, no. 1, pp. N.PAG–N.PAG, Mar. 2019, publisher: Springer Nature. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=c9cf754b-031e-3dd2-b8c7-83aad71345b1

[5] S. Greco, A. Ishizaka, M. Tasiou, and G. Torrisi, "On the Methodological Framework of Composite Indices: A Review of the Issues of Weighting, Aggregation, and Robustness," *Social Indicators Research*, vol. 141, no. 1, pp. 61–94, Jan. 2019. [Online]. Available: https://doi.org/10.1007/s11205-017-1832-9

[6] N. Imtiaz, S. Thorn, and L. Williams, "A comparative study of vulnerability reporting by software composition analysis tools," in *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 1–11. [Online]. Available: https://doi.org/10.1145/3475716.3475769

[7] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, "Structure and Evolution of Package Dependency Networks," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 102–112. [Online]. Available: https://ieeexplore.ieee.org/document/7962360

[8] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small World with High Risks: A Study of Security Threats in the npm Ecosystem," 2019, pp. 995–1010. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman

[9] L. Šubelj and M. Bajec, "Community structure of complex software systems: Analysis and applications," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 16, pp. 2968–2975, Aug. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037843711100269X

[10] A. Hora, D. Silva, M. T. Valente, and R. Robbes, "Assessing the Threat of Untracked Changes in Software Evolution," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, May 2018, pp. 1102–1113, iSSN: 1558-1225. [Online]. Available: https://ieeexplore.ieee.org/document/8453191

[11] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 181–191. [Online]. Available: https://dl.acm.org/doi/10.1145/3196398.3196401

[12] M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: a case study of the PyPI ecosystem," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 644–655. [Online]. Available: https://dl.acm.org/doi/10.1145/3236024.3236062

[13] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Softw. Engg.*, vol. 23, no. 1, pp. 384–417, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9521-5