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 present a cluster analysis of the Maven Central Repository's dependency structure to identify and assess vulnerability risks using the Goblin framework. Through analysis of over 15 million artifacts using the Leiden community detection algorithm, we identified approximately 67 thousand distinct clusters with a high modularity score. Our risk assessment framework combines CVE metrics, freshness scores, and inter-cluster connectivity patterns to evaluate cluster risk levels and potential vulnerability propagation paths. The analysis reveals that while individual clusters typically show low to moderate risk scores, the repository's highly connected structure creates critical paths for vulnerability propagation through hub clusters, some containing over 1.5 million nodes. We provide recommendations for dependency risk monitoring, including tracking of bridge nodes and prioritizing high-connectivity clusters. Our systematic approach provides a framework to identify systemic dependency risks across the repository through targeted inspections at critical points in the dependency network.**

***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 Repository (MCR), 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 MCR through a cluster-based approach, uncovering how artifacts group naturally and the risks these groupings pose.

This study utilized the Maven Central Dependency Graph (MCDG) [1], which is a Neo4j graph dataset developed to analyze the MCR. We used the dataset to explore three 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 repository?

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 Central repository?

These research questions build upon each other to address critical gaps in dependency management. Understanding natural groupings and their interactions (RQ1) is fundamental to analyzing systemic risks, as vulnerability propagation patterns follow these structural pathways. This cluster-level understanding enables targeted risk assessment (RQ2), helping prioritize security efforts across thousands of dependencies by identifying which component groups pose the greatest systemic threats. The practical implications of this analysis (RQ3) are valuable when looking to allocate security resources. This study combines community detection algorithms with multi-metric risk analysis to provide a comprehensive view of the MCR's structural and security aspects.

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 systematic analysis reveals that a few dominant clusters exert disproportionate influence, concentrating risks and creating systemic vulnerabilities. These findings demonstrate how cluster-level analysis can help more effectively target dependency management efforts by identifying critical points of vulnerability propagation in the repository.

II. Methodology and Data Collection

Our study utilizes the MCGD 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 MCR structure. The database includes detailed met-

rics 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 modeled the Maven Central Graph (MCG) in Neo4j and used the Neo4j Graph Data Science library [2] to detect communities using the Leiden algorithm. We chose Leiden for its ability to guarantee well-connected communities while maintaining high computational efficiency [3], [4]. Unlike alternatives such as the Louvain algorithm, Leiden avoids poorly connected or disconnected communities [5], [6] and ensures local optimality through continuous partition refinement.

To assess cluster risk, we employed Principal Component Analysis (PCA) to derive empirical weights for our metrics. PCA provides an objective framework for developing composite indices by identifying significant sources of variance in multidimensional data [7]–[9]. This allowed us to create unbiased risk scores that reflect the natural importance of each metric. Our analysis focused on identifying patterns and vulnerabilities within clusters, examining bridge nodes that serve as critical connectors, and assessing how vulnerable artifacts could affect others through dependency chains. This approach provided actionable insights into risk propagation and highlighted areas requiring closer monitoring.

To support reproducibility and further research, 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 MCDG into 67,669 distinct clusters, achieving a modularity score of 0.704. This modularity score indicates strong natural divisions within the dependency network. The clear community structure provides a foundation for analyzing how vulnerabilities might propagate within and between these well-defined groups.

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 in the MCR 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 repository'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 clusters 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.

Domain Analysis: Examining sample artifacts within clusters revealed notable patterns in groupings. For example, cluster 28947 demonstrated a focus on testing and monitoring frameworks, containing artifacts like *Apache FLink* testing utilities and *Graylog* log management. Similarly, cluster 30270 showed a concentration of *WSO2 Carbon* framework components, while cluster 14097 included cloud infrastructure tools such as *AWS CDK* components and *Skywalking* monitoring solutions.

This analysis suggests that while strong inter-cluster dependencies exist, as evidenced by the millions of connections between major clusters like 48568 and 29418, there are still discernible specializations within individual clusters. The balance between these specializations and the extensive inter-cluster connectivity highlights the complex nature of the MCR's dependency structure.

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

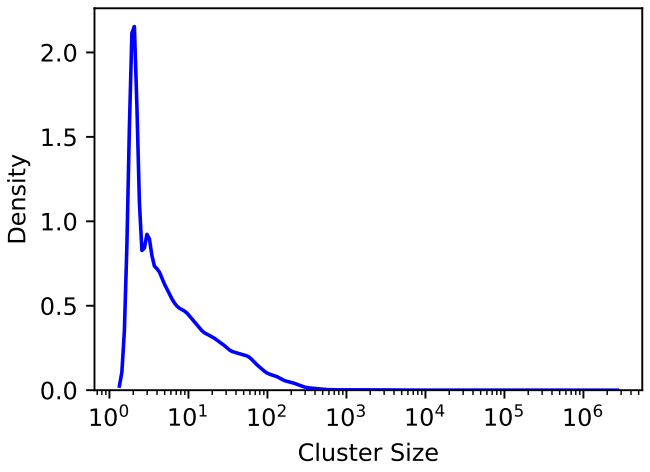

Fig. 1. Kernal Density Estimate of Cluster Sizes

Answer to 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. It is evident that most clusters have a low risk score (< 0.2). Table I contains a few select clusters from the MCDG that we wanted to highlight. The clusters are presented in order of risk score, not necessarily actual risk. Some key findings from the analysis include:

- **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 repository. Large, interconnected clusters amplify systemic risk, making them high-priority candidates for monitoring and intervention.

The analysis also revealed interesting patterns in how different metrics correlate with overall risk. High CVE counts often coincided with moderate to high severity scores, suggesting that vulnerable packages tend to have multiple serious vulnerabilities rather than isolated minor issues. Freshness scores showed a weaker correlation with CVE metrics, indicating that older packages are not necessarily more vulnerable. Popularity metrics demonstrated a moderate correlation with risk scores, supporting the intuition that widely used packages attract more security scrutiny.

Answer to 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 repository.*

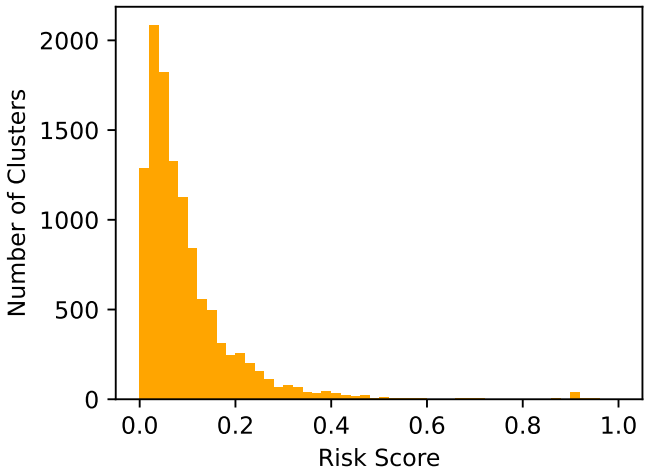

Fig. 2. Distribution of Risk Scores for Clusters

C. Potential Risks and Vulnerabilities

Our analysis revealed three key categories of risk within the MCR: 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.

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.keycloack:keycloak-core* exhibit update delays of 6-12 months, significantly increasing vulnerability exposure. This relates to findings from Imtiaz et al. [10], who emphasized the importance of continuous monitoring of vulnerability data from open-source package ecosystems and suggested that using multiple SCA tools may be necessary to ensure comprehensive vulnerability detection.

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.

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
677551	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

Risk Mitigation Recommendations: Building on Imtiaz et al.'s [10] findings regarding the effectiveness of automated vulnerability detection, we recommend:

- Implement automated dependency monitoring focused on high-risk clusters.
- Prioritize updates for bridge nodes and highly connected artifacts.
- Establish regular security audits for clusters with high risk scores.
- 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.

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

IV. THREATS TO VALIDITY

PCA Metric Weighting: While PCA provides a robust, data-driven method for determining metric weights, it introduces limitations that may impact the validity of the risk assessment model. PCA relies on the variance structure of the data, which can underemphasize metrics exhibiting low variance even if they represent substantial risk, and does not incorporate domain-specific knowledge.

Data and Algorithm Limitations: Our analysis relies on the MCDG snapshot from August 30, 2024, which may not capture the most recent ecosystem changes or dependencies hosted outside Maven Central. Additionally, the Leiden algorithm, while providing high-quality community detection, exhibits non-deterministic behavior that could produce slightly different cluster assignments, particularly for nodes at community boundaries.

V. RELATED WORK

Previous studies have explored dependency networks and community detection in software ecosystems. Kikas et al. [11] analyzed the npm ecosystem's dependency network structure, while Zimmerman et al. [12] demonstrated systemic risks from popular packages. Subelj and Bajec [13] showed community detection's effectiveness for understanding software structure, and Hora et al. [14] applied similar techniques to study API usage patterns.

Risk assessment in package ecosystems has also been studied extensively. Decan et al. [15] examined security vulnerabilities in npm packages, while Valiev et al. [16] investigated package sustainability. Our work extends these approaches by combining community detection with a comprehensive risk assessment framework that considers multiple metrics and cluster interactions.

VI. CONCLUSION

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

The composite risk assessment framework identified several vulnerabilities in the repository. 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 and large clusters with many connections as targets for vulnerabilities. Our analysis demonstrates that effective ecosystem-wide security requires a balanced approach that considers both individual package vulnerabilities and their position within the broader dependency network.

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] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, p. 026113, Feb. 2004, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.69.026113

[4] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, Feb. 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157309002841

[5] 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

[6] 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

[7] 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

[8] W. Becker, G. Caperna, M. D. Sorbo, H. Norlén, E. Papadimitriou, and M. Saisana, "COINr: An R package for developing composite indicators," *Journal of Open Source Software*, vol. 7, no. 78, p. 4567, Oct. 2022. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.04567

[9] A. M. SAVA, "WEIGHTING METHOD FOR DEVELOPING COMPOSITE INDICES. APPLICATION FOR MEASURING SECTORAL SPECIALIZATION." *Journal of Applied Quantitative Methods*, vol. 11, no. 3, Sep. 2016, publisher: Association for Development through Science & Education. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=6c469eae-dfcb-3805-b3de-66c4919220a8

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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