



Connecting the Dots: An Integrated Vulnerability Knowledge Graph for Security Practitioners

Brittany Boles; Clemente Izurieta; Ann Marie Reinhold

Accessibility Disclaimer:

For a more accessible version of this document, please submit an accessibility request form through the Montana State University Library website.

Connecting the Dots: An Integrated Vulnerability Knowledge Graph for Security Practitioners

Brittany Boles
Gianforte School of Computing
Montana State University
Bozeman, Montana
brittanyboles@msu.montana.edu

Clemente Izurieta
Gianforte School of Computing
Montana State University
Pacific Northwest National Laboratory
Idaho National Laboratory
Bozeman, Montana
clemente.izurieta@montana.edu

Ann Marie Reinhold
Gianforte School of Computing
Montana State University
Pacific Northwest National Laboratory
Bozeman, Montana
reinhold@montana.edu

Abstract—Vulnerability databases are essential to cybersecurity, providing developers with critical information about software security flaws. However, inconsistencies among vulnerability databases pose challenges for integration. To address this, we created a graph database that consolidates data from the National Vulnerability Database (NVD), GitHub Advisories, the Open Source Vulnerability (OSV) database, the Exploit Prediction Scoring System (EPSS), and the CWE-1000 View. Our graph database revealed inconsistent vulnerability severity vectors across the databases. To illustrate the utility of our graph database, we investigated how the databases reported the “top ten” most routinely exploited vulnerabilities. Our analysis revealed differences in vulnerability identifiers, and the Common Weakness Enumeration (CWE) mappings of the top ten vulnerabilities. By aggregating vulnerability information from disparate sources, this graph database supports cross-validation, increases transparency, and enables efficient complex queries.

Index Terms—Vulnerability graph database, Common Weakness Enumeration, Exploit Prediction Scoring System

I. INTRODUCTION

Cybersecurity risks in code are enumerated as vulnerabilities. Vulnerability databases catalog known security flaws. Consequently, vulnerability databases have emerged as a cornerstone of cybersecurity.

Creators and maintainers of each vulnerability database face a Sisyphean challenge. New vulnerabilities are reported daily and must be vetted for accuracy. Vulnerability submissions increased 32% from 2023 to 2024, and are estimated to continue to rise¹. The volume and velocity of new vulnerabilities being reported make maintaining the databases more difficult. Consequently, the reliability, accuracy, and completeness of the databases are impacted [1], [2].

In February 2024, the National Vulnerability Database (NVD) paused updates for several months as the National Institute of Standards and Technology (NIST) worked to address a growing backlog of submissions². The sudden disruption left the cybersecurity community scrambling for alternative

databases^{3,4}. The incident highlighted the dangers of over-reliance on a single source and emphasized the necessity for a more robust, multi-database strategy in cybersecurity.

Security tools already embrace this multi-source philosophy. Many static analysis tools aggregate multiple vulnerability databases to improve vulnerability coverage^{5,6,7}. However, because multiple databases often report the same vulnerability, the way these tools combine vulnerability data impacts which—and how many—vulnerabilities the tools report [3], [4]. Security professionals and researchers often face the challenge of receiving different vulnerability reports when analyzing the same software using static analysis tools [4]–[7]. Additionally, the number of vulnerabilities identified by static analysis tools influences how Software Quality Assurance (SQA) models evaluate system quality [8]. Discrepancies in the vulnerability databases used by these tools can lead to inaccurate assessments of system quality. Tools that treat the databases separately can report redundancy. However, aggregating vulnerability reports across databases can cause information loss, introduce errors, and reduce transparency.

Aggregation and visualization across vulnerability databases is complex due to schema differences, inconsistencies in metadata, and varying levels of abstraction. For instance, the same vulnerability may have a single identifier in one database but multiple identifiers in another. Further, vulnerability databases are constantly evolving, as are the vulnerabilities they track⁸.

Until now, no technology has enabled the analysis and visualization of vulnerabilities across vulnerability databases. Here, we created a graph database that integrates three of the most prominent vulnerability databases: the NVD, GitHub Advisories, and the Open Source Vulnerability (OSV) database. This graph database promotes cross-validation and enables the

¹<https://www.nist.gov/itl/nvd>

²<https://www.nist.gov/itl/nvd/nvd-news>

³<https://www.darkreading.com/vulnerabilities-threats/fall-of-national-vulnerability-database>

⁴<https://anchore.com/blog/navigating-the-nvd-quagmire/>

⁵<https://github.com/anchore/grype>

⁶<https://github.com/aquasecurity/trivy>

⁷<https://github.com/intel/cve-bin-tool>

⁸<https://media.blackhat.com/us-13/US-13-Martin-Buying-Into-The-Bias-Why-Vulnerability-Statistics-Suck-WP.pdf>

visualization of cliques of varying sizes while allowing for variable levels of abstraction in the underlying vulnerability databases. We improved overall vulnerability coverage and enabled security practitioners to make informed decisions based on diverse perspectives.

II. RELATED WORK

Cybersecurity researchers have explored the use of graph databases for vulnerability management. Graph databases can be advantageous for mapping vulnerabilities because they do not require a single schema, allow for the integration of diverse data sources, and support complex queries.

Bhalker et al. [9] demonstrated how a graph database can be used to visualize relationships among vulnerabilities. The researchers [9] constructed a graph database focused on relationships, such as “Breach Types”. This study advanced the use of graph databases in vulnerability analysis, but was based on a relatively small collected dataset of Cyber Security Breaches from 2009 to 2014 from varied sources.

In contrast, our graph database incorporates three extensive databases, each containing over 200,000 vulnerabilities. Our graph database interconnects vulnerabilities using both “aliased” and “related” vulnerabilities in our graphs. We also incorporate CWE, Common Vulnerability Scoring System (CVSS), and Exploit Prediction Scoring System (EPSS) data for a more holistic approach.

We are not the first to create a graph database containing the NVD. Wang et al. [10] used a knowledge graph to improve the analysis of security vulnerabilities. These researchers sourced vulnerability information from the NVD to address limitations such as poor readability and inadequate visualization of correlations between vulnerabilities. By leveraging Neo4j, they built a common vulnerabilities and exposures (CVE) knowledge graph incorporating raw data, ontology modeling, and data extraction. Their research allowed for deeper vulnerability analysis across dimensions like Common Weakness Enumerations (CWEs) and severity.

Our approach expands on previous research by incorporating a more diverse set of databases. Additionally, our graph database facilitates security analysis across different identification conventions (e.g., CVE, GitHub Security Advisory [GHSA], Ubuntu security notice[USN]).

III. METHODS

A. Database Selection

We obtained vulnerability data from the NVD, GitHub Advisories⁹, and OSV¹⁰, retrieved on March 11, 2025. We selected these vulnerability databases for their widespread adoption and integration with static analysis tools. We incorporated all three vulnerability databases into our graph database.

Unlike vendor-specific databases, which report vulnerabilities relevant only to their ecosystems, these three sources provide broad coverage. The NVD, maintained by the NIST, enriches vulnerabilities as CVEs. The GitHub Advisory Database

aggregates data from eight sources, such as GitHub security advisories, the NVD, and the npm Security Advisories database, and labels them with GHSA identifiers. The OSV integrates filtered subsets of data from 18 sources, including the GitHub Advisory Database, PyPI Advisory Database, and Go Vulnerability Database, leading to the use of multiple vulnerability identifiers. In total, the OSV has 27 different vulnerability identifier types in its database (e.g., CVE, GHSA, USN, CGA, RSEC). While GitHub Advisories and OSV are maintained by GitHub and Google respectively, both are open-source, allowing public contributions.

In our graph database, we also incorporated scores from the Exploit Prediction Scoring System (EPSS). We included EPSS to promote the evaluation of exploitability metrics concomitant with severity metrics (i.e., CVSS). EPSS assigns a score (0–1) to CVE-labeled vulnerabilities, estimating the likelihood of exploitation in the next 30 days. It also provides a percentile ranking, indicating the proportion of vulnerabilities with an equal or lower score. Integrating EPSS offers developers an additional perspective on risk assessment beyond CVSS.

We also augmented the graph with Common Weakness Enumerations (CWEs) from the CWE 1000 view¹¹. CWEs classify common software weaknesses that can lead to vulnerabilities. For example, the Log4j vulnerability (e.g., ‘CVE-2023-44228’ in the NVD and OSV, ‘GHSA-jfh8-c2jp-5v3q’ in GitHub Advisories and OSV) maps to CWE-502, which describes unsafe deserialization of untrusted data. While vulnerabilities do not always map to CWEs, many do.

B. Building The Graph Database

We integrated the vulnerabilities from the three vulnerability databases using Neo4j (version 1.61) and Python (version 3.13). All code used to build and analyze the graph database can be found at <https://doi.org/10.5281/zenodo.15177924>. Each vulnerability database was represented as a unique node type (NVD, GitHub, OSV), with each vulnerability entry corresponding to individual nodes. The attributes of these nodes were determined by the schema of the respective source databases. These attributes included information such as CVSS vectors and vulnerability descriptions.

Next, we established relationships between vulnerability nodes. The OSV schema, used by GitHub and OSV, defines an ‘alias’ and ‘related’ relationships. A ‘related’ vulnerability relationship connects two or more vulnerabilities that are (1) similar but not unique, (2) multiple similar vulnerabilities codified in the same entry, or (3) do not satisfy the strict definition of the alias. An ‘alias’ relationship connects two or more vulnerabilities that affect any software component the same way; i.e., either both vulnerabilities affect the software component or neither do¹². A patch addresses all vulnerabilities with the same alias. In our graph database, we created alias relationships for vulnerabilities with identical identifiers across the databases.

⁹<https://github.com/advisories>

¹⁰<https://osv.dev/>

¹¹<https://cwe.mitre.org/data/definitions/1000.html>

¹²<https://ossf.github.io/osv-schema/>

TABLE I
EXECUTION TIME IN MILLISECONDS FOR EXAMPLE QUERIES. QUERIES WERE PERFORMED ON UBUNTU 22.04.5 AND WITH NEO4J VERSION 1.61.

Neo4j Query	Execution Time (ms)
GitHub and OSV alias pairs MATCH (g:GitHub)-[:alias]-(o:OSV) RETURN g, o	174 ms
NVD and GitHub alias pairs published after 2018 MATCH (n:NVD)-[:alias]-(g:GitHub) WHERE datetime(n.published) > datetime('2018-01-01T00:00:00') AND datetime(g.published) > datetime('2018-01-01T00:00:00Z') RETURN n, g	2115 ms
Get top ten most routinely exploited vulnerabilities WITH ["CVE-2023-3519", "CVE-2023-4966", "CVE-2023-20198", "CVE-2023-20273", "CVE-2023-27997", "CVE-2023-34362", "CVE-2023-22515", "CVE-2021-44228", "CVE-2023-2868", "CVE-2022-47966"] AS top_vulns MATCH (v) WHERE v.id IN top_vulns RETURN v	456 ms

To add EPSS data to the ontology, we built nodes 'EPSS', where each node is the score for a CVE ID and has attributes of the EPSS score and percentile. Corresponding nodes with the CVE IDs were connected using a relationship to the EPSS node called 'epssScore'.

Each CWE in the CWE-1000 view was built into nodes, with attributes such as 'description', 'id', and 'type'. CWE nodes have relationships to vulnerability nodes via 'weakness' relationships provided by the 3 vulnerability databases. The CWE nodes also have their hierarchal mappings with levels of abstraction represented through 'parent' and 'child' relationships between the CWE nodes.

C. Analysis

Our analyses were three-pronged. First, we investigated the overlap of vulnerabilities across the three databases. In particular, we investigated discrepancies in attributes between alias nodes.

Second, we conducted a pairwise investigation for each alias relationship to determine consistency in CVSS vectors across the databases. Vectors contain manually inputted data that is used to calculate a CVSS score for a vulnerability. To promote a valid comparison, we only compared vulnerability pairs when each pair used the same version of CVSS. Different versions of CVSS are expected to generate different CVSS vectors. Therefore, we compared the CVSS vector provided by each underlying vulnerability database. If all databases had identical vulnerability data, then the CVSS vectors for all vulnerabilities that are aliases of one another would be identical.

Third, we analyzed the top ten most routinely exploited vulnerabilities in 2023 reported by DHS CISA¹³ to determine the connectedness and similarities in reporting of the top ten vulnerabilities across the databases. Note that the top ten for 2024 were not available in February 2025, when this paper was written. Our exploration of the top ten vulnerability subgraph included traversing vulnerability node relationships 'related',

¹³<https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-317a>

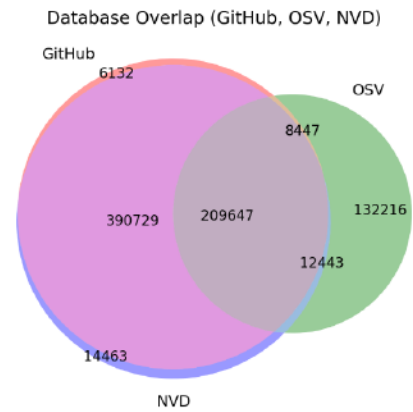


Fig. 1. Shows the distribution of the total 774,077 vulnerability nodes across databases. Overlap occurs when nodes in different databases have an alias relationship between them, implying that these are the same vulnerabilities in different databases.

'alias', 'weakness', 'epssscore' and the CWE nodes 'parent' and 'child' relationships. This exploration of the subgraph gave us a holistic view of the top ten vulnerabilities. We also tracked execution time for common queries (Table I) to demonstrate the response time of our graph database from a user perspective.

IV. RESULTS

We found an increase in total vulnerability coverage by integrating the NVD, OSV, and GitHub Advisories. Further, we found that the databases overlap in vulnerabilities (Fig. 1), allowing for cross-referencing across vulnerability databases. Such cross-referencing revealed differences in CVSS vectors for aliased vulnerabilities.

A. Vulnerability Database Relationships

Our graph database successfully integrated data from three major vulnerability databases. Each database contributed a distinct subset of the overall vulnerability landscape. In total, the NVD contributed 276,339 nodes, GitHub Advisories

TABLE II

CVSS VECTOR DIFFERENCES IN ALIAS PAIRS WITH THE SAME CVSS VERSION. "DISAGREEMENT" INDICATES THAT CVSS VECTORS DIFFERED. "TOTAL MATCHED" INDICATES MATCHING PAIRS. "NO MATCH" INDICATES ALIAS PAIRS WHERE ONLY ONE NODE USES THAT CVSS VERSION. THE "%" COLUMN SHOWS THE PERCENTAGE OF ALIAS PAIRS WITH DIFFERING CVSS VECTORS.

	Disagreement	Matched	%	No Match
CVSSv2	0	0	NA	281,817
CVSSv3	65	16,729	0.38%	65,898
CVSSv3.1	27,408	214,696	12.76%	99,882
CVSSv4	392	6,971	5.62%	22,221

contributed 268,644 vulnerability nodes, and OSV contributed 229,094 nodes for a total of 774,077 vulnerability nodes. Additionally, we had 281,448 EPSS nodes and 988 CWE nodes for a total of 1,056,513 nodes.

The NVD uses only one ID type (CVE). Similarly, GitHub only uses GHSA identifiers. Thus, the NVD has no alias or related relationships among NVD nodes; similarly, GitHub has no alias or related relationships among GHSA nodes. In contrast, the OSV database has 27 different identifiers from 18 different data sources. Therefore, the OSV database has interconnected nodes representing alias and related relationships. In total, the OSV contains 43,675 vulnerabilities with an alias and 114,882 vulnerabilities with a related vulnerability.

When comparing the three vulnerability databases, GitHub contributes 6,132 distinct vulnerabilities without aliases in the NVD or OSV databases. The NVD database contributes 14,463 distinct vulnerabilities, and the OSV database contributes 132,216 distinct vulnerabilities (Fig. 1). In total, 621,266 of the 774,077 vulnerability nodes have an alias.

Beyond aliases, our graph database nodes contained 625,439 'related' vulnerability relationships. Additionally, 672,124 nodes had an EPSS score. Vulnerabilities mapped to CWEs 1,022,272 times (as a single vulnerability can be associated with multiple CWEs). Overall, the graph database comprises 2,932,444 relationships.

B. Discrepancies in Severity Scores

Discrepancies in CVSS vectors were common. In our pairwise comparison, we identified 281,817 alias relationships where only one node had a CVSSv2 vector. Notably, there were zero instances where both nodes in a pair had CVSSv2 vectors (Table II), making the comparison of CVSSv2 vectors impossible.

There were 65,898 alias relationships in which only one node had a CVSSv3 vector, preventing direct comparison for these pairs. However, 16,729 vulnerability pairs had CVSSv3 vectors for both nodes, with 65 instances of disagreement in vectors (0.38%).

In total, 214,696 alias relationships had CVSSv3.1 vectors, with 27,408 instances of disagreement (12.76%). In total, 99,882 alias relationships were not comparable for CVSSv3.1 due to differences in how CVSS vectors were constructed.

For CVSSv4, there were 22,221 alias pairs that could not be compared due to differences in how CVSS vectors were

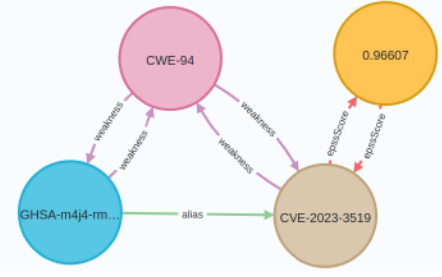


Fig. 2. Graph of the top exploited vulnerability, CVE-2023-3519. The blue GHSA node GHSA-m4j4-rmj5-w5gp represents an alias listed in GitHub Advisories, while the CVE node originates from the NVD. Both databases classify this as a CWE-94 weakness, and CVE-2023-3519 has an EPSS score of 0.96607

constructed. Among the 6,971 vulnerability pairs where both nodes had CVSSv4 vectors, 392 pairs (5.62%) had differing CVSSv4 vectors.

Severity vectors for a vulnerability often depended on the underlying vulnerability database. The discrepancies in CVSS vectors across alias pairs, specifically with version CVSSv3.1, highlight important discrepancies across the NVD, GitHub, and OSV databases. Moreover, these discrepancies indicate the subjectivity inherent in quantifying the severity of vulnerabilities, even when using the same scoring system [11].

C. Top Ten Routinely Exploited Vulnerabilities

The utility of our graph database is evident from our analysis of the database coverage of the top ten most frequently exploited vulnerabilities, their CVSS and EPSS scores, and their mappings to CWEs. The top ten vulnerabilities were reported using CVE IDs. The NVD included all ten vulnerabilities, whereas the OSV database contained only one, CVE-2021-44228. Although GitHub does not label vulnerabilities with CVE IDs, we utilized the alias relationships of the CVE nodes to identify all ten vulnerabilities under GHSA IDs in the GitHub advisories database.

Severity vectors were consistent for the top ten vulnerabilities. The CVSSv3.1 severity scores were consistent across the databases for the top ten vulnerabilities. The lowest severity score was CVE-2023-20273 at 7.2.

EPSS scores were surprisingly low for three of the ten most exploited vulnerabilities. Whereas seven of the top ten vulnerabilities had EPSS scores greater than 0.88, the remaining 3 top ten routinely exploited vulnerabilities had low EPSS scores: CVE-2023-20273 had an EPSS score of 0.07, CVE-2023-27997 had an EPSS score of 0.10651, and CVE-2023-2868 had an EPSS score of 0.07893. These low EPSS scores contradict the exploitability ranking provided by DHS CISA.

When we queried all weaknesses associated with the top ten vulnerabilities, we found that many of the top ten vulnerabilities share CWEs (Fig. 3). The most shared weakness was CWE-20, Improper Input Validation, with 4 of the top ten mapping to it.

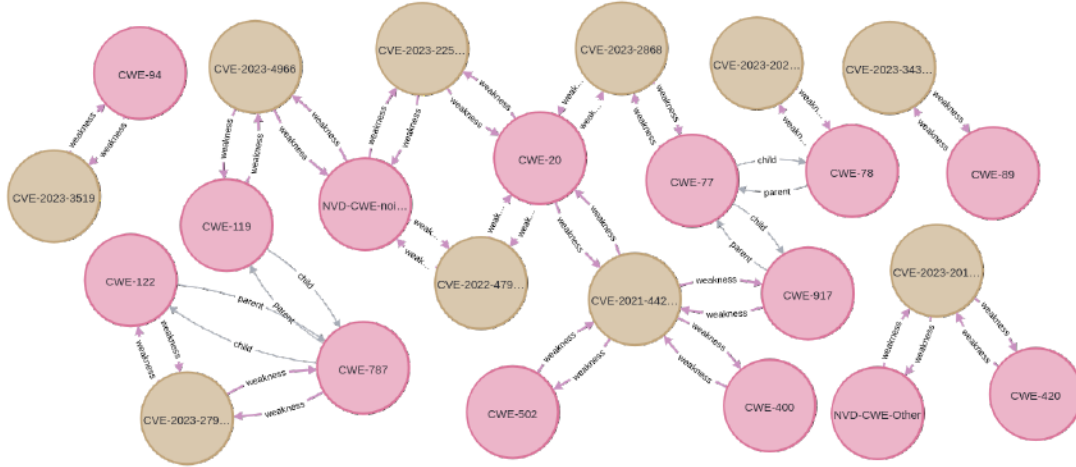


Fig. 3. Top exploited vulnerabilities in the NVD database mapped to CWEs. Here, we can see many of the top vulnerabilities and shared CWEs. The top ten vulnerabilities map to 14 CWEs. Two of those CWEs are NVD-CWE-noinfo and NVD-CWE-Other. The graph image was generated by Neo4j.

CWE mappings can be inconsistent across vulnerability databases. One illustrative mapping is for CVE-2023-4966, where the NVD maps it to CWE-199 and NVD-CWE-noinformation. Using our tool, we identified the source of the discrepancy. The NVD uses a secondary source, 'secure@citrix.com', that maps the vulnerability to CWE-199, but the NVD, as a primary source, has no information on the vulnerabilities weakness.

By including the CWE 1000 view of parent and child relationships, we added another layer of detail to our graph database. In addition, we can specify the depth of parent-child relationships to visualize. Looking at the vulnerability that mapped to the most CWEs, CVE-2021-44228, one parent-child relationship deep, we found two of the CWEs, CWE-502 and, CWE-400, mapped to the same parent CWE, CWE-664.

Looking one parent-child CWE relationship deep for the top ten vulnerabilities the resulting graph has only two path components. Nine of the top vulnerability nodes have a connected chain of parent-child relationships between their CWE nodes. Only one vulnerability, CVE-2023-20198, had no path of relationships connecting it to the other top ten vulnerabilities.

V. DISCUSSION

In this research, we proposed unifying multiple vulnerability databases—NVD, GitHub Advisories, and OSV—into a single integrated graph database. We expanded the database with additional layers of vulnerability information by including the EPSS scores and the CWE-1000 view. Although the integration of information from these databases improved our overall domain coverage, it also revealed that there are significant overlaps and inconsistencies of information between databases. The overlaps are exposed through the examination of the relationships (e.g., aliases) between the vulnerability nodes, and inconsistencies are uncovered through variations

in attribute measurements (e.g., differing CVSS scores for the same vulnerability).

Making vulnerability management faster and easier is imperative. Our analysis of the top ten most routinely exploited vulnerabilities identified by CISA reveals the utility of our integrated database solution. For example, the mapping of CWEs with corresponding CVEs revealed significant patterns in the top ten vulnerabilities, showcasing how a graph database can reveal valuable insights quickly.

A. Graph Construction and Connectivity

An integrated graph database solution offers interconnected, varied perspectives for vulnerability management. By incorporating three databases maintained by prominent organizations, GitHub, Google, and NIST, we improve overall ecosystem coverage and robustness.

In total, our database has 774,077 (Fig. 1) vulnerability nodes and 1,056,513 nodes when including EPSS and CWE nodes. The integrated graph database has 2,932,444 relationships informing users of EPSS scores, associated weaknesses, aliases, related vulnerabilities, and parent-child relationships with corresponding CWEs. Over 621,266 vulnerabilities from the combined datasets have alias relationships, allowing for cross-database comparisons. To further highlight the utility of an integrated graph database solution, this practical approach allows practitioners to easily access information with fast and easy-to-create queries.

B. Discrepancies in Cross-Database Analysis

Filtering vulnerability reports by severity is a common practice in vulnerability management [12], and practitioners often use CVSS for prioritization. A key finding from our analysis is the inconsistency in CVSS vectors reported by different databases (Table II), highlighting the subjectivity in CVSS scoring. Other researchers have reported that up to 68% of evaluators assigned different severity ratings for the

same vulnerabilities [13]. Our study demonstrates how this subjectivity propagates. While our solution does not determine which source provides the ‘correct’ severity score, it does provide key data to inform practitioner decisions.

Our tool also informs practitioner decisions via our integration of EPSS, a measure of the likelihood that a vulnerability will be exploited. However, EPSS scores only map to vulnerabilities with CVE IDs. We overcome this limitation by utilizing the ‘alias’ relationships in our graph database. By doing so, vulnerabilities with different IDs, such as GHSA or USN, can still access EPSS information. Thus, our solution allows developers the autonomy to choose the data sources that fit their needs.

C. Using the Graph Database

Using DHS CISA’s “top ten” most frequently exploited vulnerabilities, we illustrate how the relationships between vulnerabilities and their associated CWEs are suggestive of common patterns in exploit techniques. More specifically, many of the top ten vulnerabilities share common CWEs.

Nine of the top ten vulnerabilities were related via CWE parent-child relationships. This connectivity in the CWEs of the top ten vulnerabilities highlights the benefit of our integrated tool. Patterns of weakness that are commonly exploited emerge without the need to scan multiple vulnerability databases or click one’s way through the CWE-1000 view.

VI. CONCLUSION

Our database provides a computational solution that decreases cognitive workloads and the difficulties of compiling vulnerability information manually. Moreover, query execution time is fast despite our graph database containing over a million nodes and nearly three million relationships (Table 1). Thus, by integrating multiple vulnerability databases into a single graph database, we provide an elegant and efficient solution for the complex problem of vulnerability mapping and management.

Our graph database also serves as a valuable resource for vulnerability database maintainers and security practitioners. By making inconsistencies—such as different vulnerability identifiers or divergent CWE mappings—readily visible through graph relationships, the database enables maintainers to evaluate and improve their data quality. While our current solution does not resolve all inconsistencies automatically, it empowers the database maintainers and users to make informed decisions when confronted with conflicting information.

Our future work will focus on expanding data sources, incorporating real-time updates, and adding features for uploading static analysis tool reports into our graph database. By strengthening this solution, we aim to further support security practitioners in mitigating threats with greater accuracy and efficiency.

ACKNOWLEDGMENT

We thank Gage Nesbit, Connor Munson, Eric O’Donoghue, Madison Munro, and Ryan Cummings for their contributions

to this article. This research was conducted with support from the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract 70RSAT22CB0000005. Any opinions contained herein are those of the author and do not necessarily reflect those of DHS S&T.

REFERENCES

- [1] R. Croft, M. A. Babar, and M. M. Kholoosi, “Data quality for software vulnerability datasets,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 121–133.
- [2] Y. Jiang, M. Jeusfeld, and J. Ding, “Evaluating the data inconsistency of open-source vulnerability repositories,” in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ser. ARES ’21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: <https://doi.org/10.1145/3465481.3470093>
- [3] B. Boles, E. O’Donoghue, A. Redempta Manzi Muneza, G. Perkins, C. Izurieta, and A. Marie Reinhold, “Deciphering Discrepancies: A Comparative Analysis of Docker Image Security,” in *2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2024, pp. 254–259. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/SCAM63643.2024.00034>
- [4] A. M. Reinhold, B. Boles, A. R. M. Muneza, T. McElroy, and C. Izurieta, “Surmounting challenges in aggregating results from static analysis tools,” *Military Cyber Affairs*, vol. 7, 2024. [Online]. Available: <https://digitalcommons.usf.edu/cgi/viewcontent.cgi?article=1101&context=mca>
- [5] A. M. Reinhold, T. Weber, C. Lemak, D. Reimanis, and C. Izurieta, “New version, new answer: Investigating cybersecurity static-analysis tool findings,” in *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2023, pp. 28–35.
- [6] E. O’Donoghue, A. M. Reinhold, and C. Izurieta, “Assessing security risks of software supply chains using software bill of materials,” in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*, 2024, pp. 134–140.
- [7] E. O’Donoghue, B. Boles, C. Izurieta, and A. M. Reinhold, “Impacts of software bill of materials (sbom) generation on vulnerability detection,” in *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 67–76. [Online]. Available: <https://doi.org/10.1145/3689944.3696164>
- [8] C. Izurieta, D. Reimanis, E. O’Donoghue, K. Liyanage, A. R. M. Muneza, B. Whitaker, and A. M. Reinhold, “A generalized approach to the operationalization of software quality models,” *PeerJ Computer Science*, vol. 10, p. e2357, 2024. [Online]. Available: <http://doi.org/10.7717/peerj-cs.2357>
- [9] P. M. Bhalekar and J. R. Saini, “Comprehensive exploration of the role of graph databases like neo4j in cyber security,” in *2024 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2024, pp. 1–4.
- [10] Y. Wang, Y. Zhou, X. Zou, Q. Miao, and W. Wang, “The analysis method of security vulnerability based on the knowledge graph,” in *Proceedings of the 2020 10th International Conference on Communication and Network Security*, ser. ICCNS ’20. New York, NY, USA: Association for Computing Machinery, 2021, p. 135–145. [Online]. Available: <https://doi.org/10.1145/3442520.3442535>
- [11] R. Wang, L. Gao, Q. Sun, and D. Sun, “An improved cvss-based vulnerability scoring mechanism,” in *2011 Third International Conference on Multimedia Information Networking and Security*, 2011, pp. 352–355.
- [12] Z. D. Wadhams, C. Izurieta, and A. M. Reinhold, “Barriers to using static application security testing (sast) tools: A literature review,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ser. ASEW ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 161–166. [Online]. Available: <https://doi.org/10.1145/3691621.3694947>
- [13] J. Wunder, A. Kurtz, C. Eichenmüller, F. Gassmann, and Z. Benenson, “Shedding light on cvss scoring inconsistencies: A user-centric study on evaluating widespread security vulnerabilities,” in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 1102–1121.