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

1st Brittany Boles
*Gianforte School of Computing*
*Montana State University*
Bozeman, Montana
brittanyboles@msu.montana.edu

2nd Clemente Izurieta
*Gianforte School of Computing*
*Montana State University*
*Pacific Northwest National Laboratory*
*Idaho National Laboratory*
Bozeman, Montana
clemente.izurieta@montana.edu

3rd 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 GitHub Advisories, the Open Source Vulnerability (OSV) database, the National Vulnerability Database (NVD), the Exploit Prediction Scoring System (EPSS), and the CWE-1000 View. Our graph database revealed inconsistent vulnerability severity score 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 database, Graph database, Common Vulnerabilities and Exposures, Common Weakness Enumeration, Exploit Prediction Scoring System**

## I. Introduction

Cybersecurity risks in code are enumerated as vulnerabilities. Vulnerability databases catalogue 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 rise[1]. The volume and velocity of new vulnerabilities being reported, makes 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 submissions[2]. 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 will often report the same vulnerability, the way these tools combine vulnerability data impacts which—and how many—vulnerabilities the tools report [3], [4]. Keeping the databases separate can lead redundant reportings of the same vulnerability. However, aggregating a vulnerabilities reports across databases can cause information loss, reduce transparency, and can be problematic when vulnerability databases disagree.

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 track[8].

Until now, no technology has enabled analysis and visualization of vulnerabilities across vulnerability databases. Here, we created a graph database that integrates three of the most prominent vulnerability databases: GitHub Advisories, the Open Source Vulnerability (OSV) database, and the NVD. This graph database promotes cross-validation and enables the visualization of cliques of varying sizes, while allowing for variable levels of abstraction in the underlying vulnerability databases. We improved overall vulnerability coverage, and enable security practitioners to make informed decisions based on diverse perspectives.

[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

## II. Related Work

Cybersecurity researchers have explored the use of graph databases for vulnerability management. A graph database can be advantageous for mapping vulnerabilities because it does not require a single schema. Moreover, graph databases allow for integration of diverse data sources and support complex queries (with multiple joins based on attributes such as severity, ecosystem, and software version).

Bhalker et al. [5]demonstrated how a graph database can be used to visualize relationships among vulnerabilities. The researchers [5] 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. [6] 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 GitHub Advisories[9], OSV[10], and the NVD[11], retrieved on March 11, 2025. These databases were selected for their widespread adoption and integration with static analysis tools[12,13,14].

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 NVD is managed by MITRE and sponsored by the U.S. Department of Homeland Security (DHS) and the Cybersecurity and Infrastructure Security Agency (CISA). 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 it's 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 enable 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 view[15]. 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 database using Neo4j (version 1.61) and Python (version 3.13). All code used to build and analyze the graph database can be found at (LINK REPO) Each vulnerability database was represented as a unique node type (GitHub, OSV, NVD) 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 scores and vulnerability descriptions.

Next, we established relationships between vulnerability nodes. The OSV schema, used by GitHub and OSV, define an 'alias' and 'related' relationship.

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) does not satisfy the strict definition of 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 do[16]. A patch addresses all vulnerabilities with the same

[9]https://github.com/advisories
[10]https://osv.dev/
[11]https://nvd.nist.gov/vuln/
[12]https://github.com/anchore/grype
[13]https://github.com/aquasecurity/trivy
[14]https://github.com/intel/cve-bin-tool
[15]https://cwe.mitre.org/data/definitions/1000.html
[16]https://ossf.github.io/osv-schema/

TABLE I
EXECUTION TIME IN MILLISECONDS FOR NEO4J EXAMPLE QUERIES. DEMONSTRATES EXAMPLES OF MULTI JOINT SEARCHES TO OUR GRAPH DATABASE.

| Neo4j Query | Execution Time (s) |
|---|---|
| **Overlap in vulnerabilities between the OSV and GitHub**<br>`MATCH (g:GitHub)-[:alias]-(o:OSV)`<br>`RETURN g, o` | 174ms |
| **Collects nodes from the NVD and GitHub when they are aliases and both were published after 2018.**<br>`MATCH (n:NVD)-[:alias]-(g:GitHub)`<br>`WHERE datetime(n.published) > datetime('2018-01-01T00:00:00')`<br>`AND datetime(g.published) > datetime('2018-01-01T00:00:00Z')`<br>`RETURN n, g` | 2115ms |
| **Get top ten most routinely exploited vulnerabilities**<br>`WITH ["CVE-2023-3519", "CVE-2023-4966", "CVE-2023-20198","CVE-2023-20273",`<br>`"CVE-2023-27997","CVE-2023-34362", "CVE-2023-22515","CVE-2021-44228",`<br>`"CVE-2023-2868", "CVE-2022-47966"] AS top_vulns`<br>`MATCH (v) WHERE v.id IN TopVulns`<br>`RETURN v` | 456ms |

alias (and no others). In our graph database, we created alias relationships for vulnerabilities with identical identifiers across the databases.

To add EPSS data to the ontology we built nodes 'EPSS' where each node is the score from a CVE ID and has attributes of the EPSS score, including the percentile. Corresponding nodes with a CVE ID 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 a 'weakness' relationship. The 'weakness' relationship details are provided by the three vulnerability databases. The CWE nodes also have their own hierarchal mappings with levels of abstraction represented through 'child' and 'parent' 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. Recall that an alias nodes are two or more representations of a single vulnerability.

Second, we conducted a pairwise investigation for each alias relationship to determine consistency in CVSS scores across the databases. The range of CVSS scores is from 0-10, where 10 is the greatest severity of a vulnerability. To ensure a valid comparison we only compared the pairs when they both used the same version of CVSS. Different versions of CVSS will generate different CVSS vectors. We compared the CVSS vector provided by each underlying vulnerability database. Each pair was thus examined for differences in CVSS vectors.

Third, we analyzed the top ten most routinely exploited vulnerabilities in 2023 reported by the CISA[17] 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. We also tracked execution time for common queries (Table I) to demonstrate the response time of our graph database from a user perspective.

[17]https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-317a

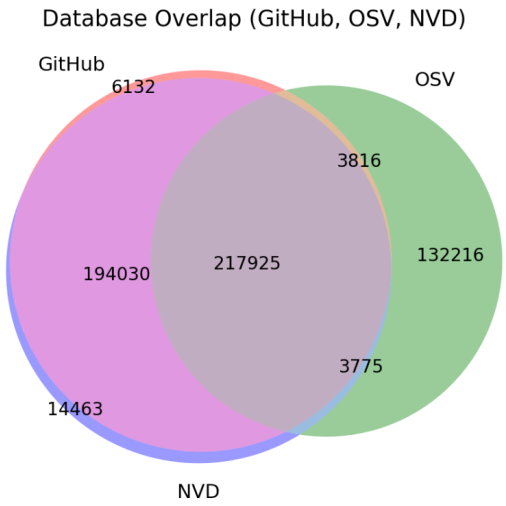


Fig. 1. Shows the distribution across database of the total 774,077 vulnerability nodes. Overlap occurs when two nodes in different databases have an alias relationship between them –implying these are the same vulnerability in different database.

### D. Graph Construction and Connectivity

## IV. RESULTS

We found an increase in total vulnerability coverage by integrating the NVD, OSV and GitHub Advisories. Further we found that the databases significantly overlap in vulnerabilities allowing for cross-referencing. When comparing alias vulnerabilities across databases we found significant difference in CVSS vectors. Finally we investigated the top ten most routinely exploited vulnerabilities. Our exploration of the top ten vulnerability subgraph included traversing vulnerability node relationships 'related', 'alias', 'weakness', 'epssscore' and the CWE nodes 'child' and 'parent' relationships. This

TABLE II
WE COMPARE SEVERITY SCORES ACROSS DATABASES USING ALIAS RELATIONSHIPS. FOR ALIAS PAIRS WITH THE SAME CVSS METRIC VERSION, WE ANALYZE SCORE DIFFERENCES. "TOTAL MATCHED" COUNTS SUCH PAIRS, WHILE "NO MATCH" INDICATES 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% | 65898 |
| **CVSSV3.1** | 27,408 | 214,696 | 12.76% | 99,882 |
| **CVSSV4** | 392 | 6,971 | 5.62% | 22,221 |

exploration of the subgraph gave us a holistic view of the top ten vulnerabilities.

### A. Vulnerability Database Relationships

The vulnerability graph database successfully integrated data from three major databases. Each database contributed a distinct subset of the overall vulnerability landscape. In total GitHub Advisories contributed 268,644 vulnerability nodes, OSV contributed 229,094 nodes and the NVD contributed 276,339 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), thus the NVD has zero alias or related relationships between NVD nodes. Similarly GitHub only uses GHSA identifiers, and consequently has no interconnected nodes. In contrast OSV database has 27 different identifiers from different 18 data sources. Therefore OSV nodes can have 'alias' and 'related' relationships interconnecting themselves. In total the OSV has 43,675 vulnerabilities with an alias and 114882 vulnerabilities with a related vulnerability all also in the OSV.

When comparing the three vulnerabilities databases against each other we find GitHub contributes 6,132 unique vulnerabilities without an aliases in the NVD or OSV databases. NVD contributes 14,463 unique vulnerabilities. OSV contributes 132,216 vulnerabilities with out overlap to the NVD or GitHub Advisories (Fig. 1). In total 621,266 out of the 774,077 vulnerability nodes have an alias.

Aliases were not the only relationship we included. 625,439 'related' relationships connected our database nodes. Moreover, 672,124 nodes had an EPSS score and vulnerabilities mapped to CWEs 1,022,272 times (vulnerabilities can map to multiple CWEs). In total the graph databases has 2,932,444 relationships.

### B. Discrepancies in Severity Scores

There were 281,817 pairs where only one node had a CVSSV2 vector, making severity comparison impossible. Notably, there were zero instances where both nodes in a pair had CVSSV2 vectors.(Table II)

There were 65,898 pairs where only one node had a CVSSV3 vector, preventing direct comparison. However, 16,729 vulnerability pairs had CVSSV3 vectors for both nodes, with 65 instances of disagreement (0.38%).


Fig. 2. Top exploited vulnerability CVE-2023-3519 graph. Blue GHSA node is an aliases that is contained in GitHub Advisories. CVE node is from the NVD. Both databases state this is a CWE-78 weakness.

In total, 214,696 pairs had CVSSV31 vectors, with 27,408 instances of disagreement (12.76%). Additionally, 99,882 pairs were not comparable due to version mismatches.

There were 22,221 pairs that couldn't be compared due to version differences. Among 6,971 vulnerability pairs where both nodes had CVSSV4 vectors, 392 pairs (5.62%) had differing CVSSV4 scores.

Severity scores for a vulnerability can depend on the source used. The discrepancies in CVSS scores specifically with version CVSSV3.1 highlight the subjectivity of quantifying vulnerabilities [7], and the variations of answers across databases.

### C. Top Ten Routinely Exploited Vulnerabilities

We demonstrate the utility of our graph by analyzing the top ten most frequently exploited vulnerabilities. Our analysis includes the database coverage of these vulnerabilities, their CVSS and EPSS scores, as well as their mappings to CWEs.

The top ten vulnerabilities were reported using CVE IDs. Our findings NVD included all of the vulnerabilities, while the OSV database only contained 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.

The CVSSV3.1 severity scores were consistent across the database for the top ten vulnerabilities. The lowest severity score was from CVE-2023-20273 at 7.2. Seven of the top ten vulnerabilities had EPSS scores greater than 0.88. However, the other 3 of the 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.

We queried all the weakness the top ten vulnerabilities mapped to, and any alias connected nodes. Many of the top vulnerabilities share CWEs (Fig. 3). The most common weakness was CWE-20, Improper Input Validation, with 4 of the top ten mapping to it. One interesting mapping is with CVE-2023-4966 where the NVD maps it to CWE-199 and NVD-CWE-noinformation. Upon further investigation we

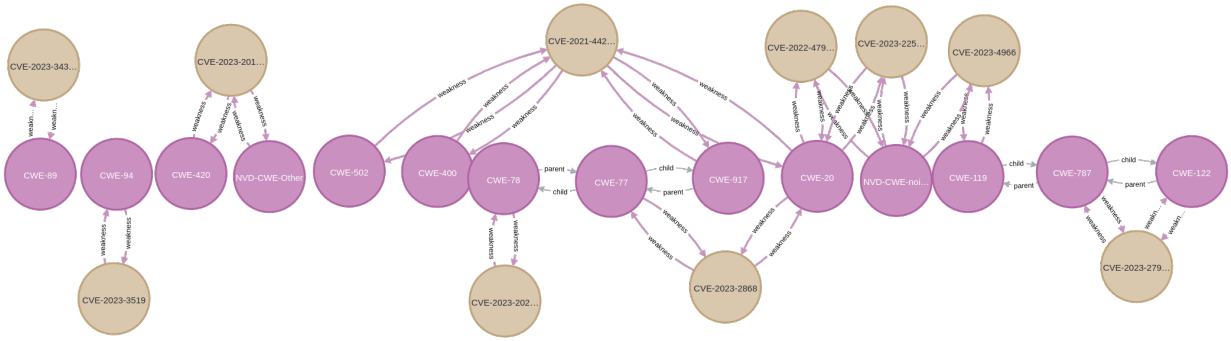


Fig. 3. Top exploited vulnerabilities in the NVD mapped to CWEs. Here we can see many of the top vulnerabilities share CWEs. The top ten vulnerabilities map to 14 CWEs. Two of those CWEs are NVD-CWE-noinfo and NVD-CWE-Other

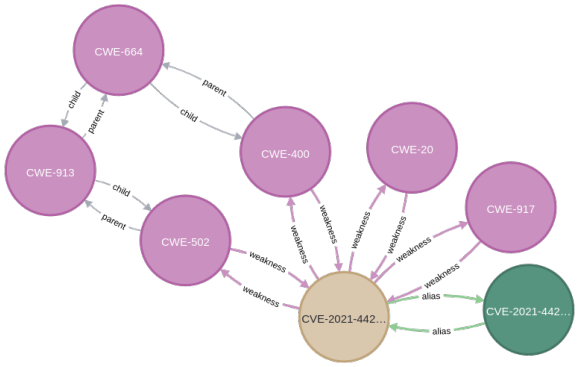


Fig. 4. CVE-2013-44228 graph with CWE relationships. The Green node is sourced from OSV and tan from the NVD.

know this occurs because the NVD using a secondary source 'secure@citrix.com' maps the id to CWE-199 but the NVD as a primary source has no information on what weakness the id maps to.

When including the CWE 1000 view child and parent relationships we get another layer of detail for these top exploited vulnerabilities. When looking at the vulnerability that mapped to the most CWEs, CVE-2021-44228 we found two of the CWEs map to the same parent CWE, CWE-664 (Fig. 4). If we zoom out and look at all the top ten vulnerabilities and the subgraph of CWEs related to them, the resulting graph has only two path components. Nine of top vulnerability nodes all have some chain of child parent nodes 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 work, we proposed integrating multiple vulnerability databases, GitHub Advisories, OSV, and NVD, into a unified graph database. We added additional layers of vulnerability knowledge by including the EPSS database and CWE-1000 view. The integration of these databases improved our overall domain coverage. While the overall coverage was increased we also found significant overlap between databases. We represented the overlap through relationships between the vulnerability nodes. These relationships enable cross-referencing information within milliseconds. Cross-referencing the vulnerability databases revealed varying CVSS scores.

Our analysis the top ten routinely exploited vulnerabilities identified by CISA showcased our graph utility. While the NVD and GitHub Advisories contained all ten vulnerabilities, OSV included only CVE-2021-44228. Mapping CWEs revealed major patterns in the top ten vulnerabilities, demonstrating how a graph database can uncover deep insights with minimal effort and time required. We state that a critical part of securing systems is making is making vulnerability management fast and easy for developers.

### A. Graph Construction and Connectivity

Our graph database offers interconnected varied perspectives for vulnerability management. By incorporating three databases maintained by prominent organizations, GitHub, Google and the NIST we improve overall ecosystem coverage and robustness. In total our database has 774,077 (Fig. 1) vulnerability nodes and in total 1,056,513 nodes when including EPSS and CWE nodes. We have 2,932,44 relationships informing users of EPSS scores, associated weaknesses, alias or related vulnerabilities and child-parent relationships of CWEs. Over 621,266 vulnerabilities from the combined datasets have alias relationships, allowing for cross-database comparisons. We present interconnected information from expert perspectives, all easily accessible with fast queries to one source.

### B. Discrepancies Found in Cross-Databases Analysis

One of the most striking findings from our analysis is the inconsistency in CVSS scores across databases. Filtering a vulnerability report by severities is a common practice in vulnerability management [8]. The Most severe vulnerabilities are typically prioritized. Our analysis of CVSS vectors revealed that for 12.76% of the CVSSV3.1 pairs, there were discrepancies, highlighting the subjective nature of severity assessments. Previous research has shown that CVSS scores are subjective, with up to 68% of evaluators assigning different severity ratings for the same vulnerabilities, highlighting inconsistencies in the scoring process [9]

Our database doesn't inform users which source provided the 'correct' severity score. Merely we aim to give users the ability to choose which source the trust. We also assert multiple perspectives can enable more informed decisions.

Furthermore, integrating EPSS scores provides an additional perspective. EPSS offers an empirical measure of the likelihood that a vulnerability will be exploited but only maps to vulnerabilities with CVE ids. Utilizing the 'alias' relationships in our graph database, vulnerabilities with different ids such as GHSA, or USN can still have this extra layer of information. Once again this gives developers all the information, and the autonomy to choose the sources that fit their needs with out the time and effort of scouring the internet and aggregating multiple sources.

### C. Using the Graph Database

Using the relationships between our nodes we were able to find the top ten routinely exploited vulnerabilities in GitHub Advisories. The OSV only contained the GHSA and CVE identifiers for the Log4j vulnerability in the top ten vulnerabilities.

The relationships between these vulnerabilities and their associated CWEs reveal common patterns in exploit techniques. We found many of the top ten vulnerabilities to share CWEs. When including the CWE child parent relationships we found 9 out of the ten top vulnerabilities weaknesses, CWEs, were related. This connectivity between the CWEs of the top ten most routinely exploited vulnerabilities highlights the benefit of this integrated view. Patterns of weakness being commonly exploited emerge with out the need for scouring multiple vulnerability databases and clicking your way through the CWE-1000's view.

## VI. Conclusion

In this work, we developed an integrated graph database combining OSV, GitHub Advisories, and NVD to enhance vulnerability management. Linking data across these sources expanded vulnerability coverage and enabled us to identified significant overlap. Additionally, the database was enriched with the EPSS database and the CWE 1000 view to offer diverse perspectives for vulnerability management. Discrepancies in CVSS scores and CWE mappings were found, emphasizing the value of multi-source intelligence for security professionals.

The ability to cross-reference vulnerabilities enables informed decisions making around security risks. Incorporating vulnerability relationships, the CWE 1000 view and EPSS scores, enables a robust and in-depth analysis of the top ten most routinely exploited vulnerabilities. This integrated data source approach empowers security teams to make more informed decisions based on diverse perspectives. We have demonstrated numerous practical queries to our database, all of which take mere seconds. Our graph database allows not only searches across sources but also multi-joint searches. These benefits can reduce the workload for security practitioners and make securing software easier.

Our future work will focus on expanding data sources, incorporating real-time updates, and providing an API for accessibility. By strengthening this system, we aim to further support security practitioners in mitigating threats with greater accuracy and efficiency.

## Acknowledgment

We would like to thank Gage Nesbit for his contributions to the graph database, Eric O'Donogue for his table beautification skills and Madison Munro for their feedback. 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=1101context=mca

[5] 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.

[6] 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

[7] 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.

[8] 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

[9] 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.