A Compact Vulnerability Knowledge Graph for Risk Assessment

JIAO YIN, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Australia
WEI HONG, School of Artificial Intelligence, Chongqing University of Arts and Sciences, Chongqing, China
HUA WANG, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Australia
JINLI CAO, Department of Computer Science and Information Technology, La Trobe University, Melbourne, Australia
YUAN MIAO and YANCHUN ZHANG, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Australia

Software vulnerabilities, also known as flaws, bugs or weaknesses, are common in modern information systems, putting critical data of organizations and individuals at cyber risk. Due to the scarcity of resources, initial risk assessment is becoming a necessary step to prioritize vulnerabilities and make better decisions on remediation, mitigation, and patching. Datasets containing historical vulnerability information are crucial digital assets to enable AI-based risk assessments. However, existing datasets focus on collecting information on individual vulnerabilities while simply storing them in relational databases, disregarding their structural connections. This article constructs a compact vulnerability knowledge graph, VulKG, containing over 276 K nodes and 1 M relationships to represent the connections between vulnerabilities, exploits, affected products, vendors, referred domain names, and more. We provide a detailed analysis of VulKG modeling and construction, demonstrating VulKG-based query and reasoning, and providing a use case of applying VulKG to a vulnerability risk assessment task, i.e., co-exploitation behavior discovery. Experimental results demonstrate the value of graph connections in vulnerability risk assessment tasks. VulKG offers exciting opportunities for more novel and significant research in areas related to vulnerability risk assessment. The data and codes of this article are available at https://github.com/happyResearcher/VulKG.git.

CCS Concepts: • **Security and privacy → Vulnerability management**; • **Information systems → Entity relationship models**;

Additional Key Words and Phrases: Knowledge graph, vulnerability risk assessment, vulnerability co-exploitation, link prediction

Authors' Contact Information: Jiao Yin (Corresponding author), Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Victoria, Australia; e-mail: jiao.yin@vu.edu.au; Wei Hong, School of Artificial Intelligence, Chongqing University of Arts and Sciences, Chongqing, China; e-mail: hongwei.auto@outlook.com; Hua Wang, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Victoria, Australia; e-mail: hua.wang@vu.edu.au; Jinli Cao, Department of Computer Science and Information Technology, La Trobe University, Melbourne, Victoria, Australia; e-mail: j.cao@latrobe.edu.au; Yuan Miao, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Victoria, Australia; e-mail: yuan.miao@vu.edu.au; Yanchun Zhang, Institute for Sustainable Industries and Liveable Cities, Victoria University, Melbourne, Victoria, Australia; e-mail: yanchun.zhang@vu.edu.au.

ACM 1556-472X/2024/7-ART194
https://doi.org/10.1145/3671005

ACM Reference format:
Jiao Yin, Wei Hong, Hua Wang, Jinli Cao, Yuan Miao, Yanchun Zhang. 2024. A Compact Vulnerability Knowledge Graph for Risk Assessment. *ACM Trans. Knowl. Discov. Data.* 18, 8, Article 194 (July 2024), 17 pages.
https://doi.org/10.1145/3671005

1 Introduction

Software vulnerabilities put critical data of organizations and individuals under cyber threat. Historically, examples of software vulnerabilities imposing significant damage to end users and vendors are not rare to see. Just in 2022, a vulnerability numbered CVE-2022-22965 linked to Spring4Shell made many headlines since it was used in nearly everything from gaming systems and consumer platforms to critical digital infrastructure.[1]

Vulnerability risk assessment and prioritization aim to identify potential software weaknesses and rank them according to their severity and impact. Based on the results of vulnerability assessment, remediation or mitigation can be prioritized to protect the most critical assets when the cyber security budgets and resources are limited. Therefore, assessment and prioritization for disclosed vulnerability play a vital role in protecting information systems and networks from cyberattacks, intrusions, malware, and various types of data breaches [18].

In industry, the **Common Vulnerability Scoring System (CVSS)** serves as the de facto industry standard for vulnerability assessment. Although widely adopted, CVSS has constantly been questioned due to its simple scoring approach and heavy reliance on expert knowledge [2, 33]. In academia, researchers tried to discover and assess vulnerability risks from public information [20, 33], including descriptions, CVSS metrics, social media streams, etc. Neural language models and state-of-the-art machine learning and deep learning algorithms are leveraged to mine the latent knowledge behind a large variety of raw data and make informed decisions [8, 15, 34]. However, the performance of these studies relies heavily on the availability and richness of disclosed vulnerability and its exploit records.

Currently, well-known and publicly accessible data sources for vulnerability risk assessment include **National Vulnerability Database (NVD)**,[2] Chinese NVD,[3] **Exploit Database (EDB)**,[4] **Common Vulnerabilities and Exposures (CVE)** Details,[5] **Common Weakness Enumeration (CWE)** database,[6] CVE,[7] etc. Although multiple data sources exist for mining, they only contain unstructured or semi-structured raw data on vulnerabilities and exploits. Researchers have to start from scratch in terms of data collection, selection, preprocessing, and integration to prepare standard datasets for algorithm development, which inevitably consumes effort that could have been directed toward algorithm design. Additionally, due to different data collection and processing methods, researchers can hardly compare their algorithms with existing works directly or reproduce others' works painlessly.

Furthermore, all aforementioned data sources are organized in a relational and preliminary way. Therefore, previous data-driven vulnerability risk assessment works mainly focused on feature extraction and representation from individual vulnerabilities, ignoring structural information and relationships across different vulnerabilities and related contexts [33, 34]. Additionally, some latent

[1] https://nvd.nist.gov/vuln/detail/cve-2022-22965
[2] https://nvd.nist.gov/
[3] https://www.cnvd.org.cn/
[4] https://www.exploit-db.com/
[5] https://www.cvedetails.com/
[6] https://cwe.mitre.org/
[7] https://www.cve.org/

topological connections between different data sources have yet to be fully utilized in this research domain.

Knowledge Graph (KG), as a new data structure containing not only the topological connectivity information between nodes but also the non-topological attributes and features of nodes and relationships, has emerged as the main tool for knowledge representation and reasoning [16, 37, 39]. It is reasonable to expect that an integrated software **Vulnerability KG (VulKG)** might improve the data quality for current research in vulnerability assessment and prioritization [8, 36].

This article constructs a formatted and processed vulnerability KG named VulKG for vulnerability risk assessment and prioritization tasks. We collect, process, and integrate multimodal data from various sources into VulKG, which is organized and stored in a graph database manner. Specifically, a use case on VulKG verifies the effectiveness of structural information in vulnerability risk assessment tasks. The contributions of this article are summarized as follows.

- —We construct a compact VulKG for software vulnerability assessment, containing over 276,000 nodes and 1 million relationships representing the connections between vulnerabilities, exploits, affected products, vendors, referred domain names, and more. VulKG can serve as the knowledge base for vulnerability assessment-related question-and-answer systems and vulnerability risk assessment tasks.
- —We provide a use case of leveraging VulKG to address The **Vulnerability Co-Exploitation Behavior Discovery (VCBD)** task, which involves identifying whether two vulnerabilities can be exploited in combination to cause a significant security breach. We systematically investigate the representational power of non-topological features extracted from nodes' properties and topological features extracted from graph structures through rigorous experiments. Our findings confirm that the topological features extracted from graph structures, in combination with nodes' properties, are highly effective in identifying vulnerability co-exploitation behavior.
- —We implement VulKG on the Neo4j graph database platform, and share the Cypher codes for VulKG deployment on GitHub. This makes it easy for other researchers to refactor and tailor VulKG into subgraphs using Cypher query languages. We also share the codes of the VCBD use case to inspire more customized vulnerability risk assessment tasks driven by graph intelligence.

The rest of this article is organized as follows. Section 2 presents related works on software VulKGs and vulnerability risk assessment. Section 3 describes the process of designing, constructing and implementing VulKG, followed by the vulnerability co-exploitation behavior discovery use case in 4. We also analyze the influence of topological and non-topological features on the VCBD task, and report the benchmark performance achieved on VCBD in Section 4. Finally, Section 5 summarizes this work with a limitation analysis and outlook for future work.

2 Related Works

2.1 Software Vulnerability Risk Assessment

When considering the term vulnerability assessment, there are two categories based on the phase of the vulnerability lifecycle during which the assessment is performed. Before being discovered, the primary goal is to discover the inherent risks at the code and architecture level. This is referred to as vulnerability discovery [7]. However, due to limited resources and the increasing number of vulnerabilities, after being discovered and published, the primary goal changes to prioritizing vulnerabilities to reduce losses [18]. This is called disclosed vulnerability assessment, which includes different aspects such as exploitation, impact, severity, type, and others [18].

The CVSS is the de facto industry standard for vulnerability assessment, but it has been constantly questioned due to its simple scoring approach and heavy reliance on expert knowledge. To counter these limitations, researchers have increasingly resorted to data-driven approaches, leveraging rising deep learning techniques [19, 21]. Among these approaches, an increasing number of scholars have started to mine the description text of vulnerabilities, incorporating other contextual information from vendors, end-users, and even social media for exploitation analysis [2, 6, 10, 19, 24, 28]. This significantly reduces the requirement for domain-specific knowledge and eases time delay problems in CVSS.

Vulnerability Exploitability Prediction (VEP) is a typical vulnerability risk assessment task to predict whether a vulnerability will be exploited. The common practice is to extract features from public vulnerability information, such as descriptions, gained access types, platforms and affected products, and then build binary classification models for VEP [34]. Bozorgi et al. [2] extracted one-hot encoded text features on a predefined tokens list. Other text feature extraction methods for VEP included statistical features [25], word embeddings [28], and ExBERT model [33]. In particular, [34] proposed an online learning framework for VEP to overcome the possible concept drift problem.

To prioritize exploitable vulnerabilities, another important vulnerability risk assessment task is **Vulnerability Exploitation Time Prediction (VETP)**, which predicts a vulnerability's possible exploitation time. Jacobs et al. [11] tried to speculate how a vulnerability could possibly be exploited within the first twelve months after disclosure. With the help of tweet discussion data, [4] are among the first ones attempting to conduct time prediction in month and day granularity. They also designed a system for vulnerability exploit scoring and timing based on that [3]. AlEroud and Karabatis [1] proposed a contextual misuse method combined with an anomaly detection technique to detect zero-day cyber-attacks. Yin et al. [35] formulated VETP as a multi-class classification problem, and vulnerabilities were classified into three classes based on whether they were exploited before and after the disclosure or on the same day of disclosure.

In summary, the aforementioned works on vulnerability risk assessment tasks focus on individual vulnerability features and ignore structural connections.

2.2 Graph-Based Software Vulnerability Assessment

Considering the scattered status of the traditional vulnerability-related data sources, researchers have started introducing graph approaches in recent years to integrate them [17]. Noel et al. [22] offered an industry-level application called CyGraph with a graph-based cybersecurity model, integrating isolated data from the comprehensive source, and crafted a domain-specific query language called CyGraph Query Language. While [12, 27] focused more on how to identify entities and relations to construct a security KG, [23, 26, 29, 31, 38] tried to build a static VulKG from widely used data sources such as CVE, NVD, CWE, NNVD, and CPE. Du et al. [5] tried to match CVE information with Java component metadata in the Maven repository and project collaboration data on GitHub. They offered a solution to improve traceability links prediction between vulnerability and software components. Kiesling et al. [13] constructed an evolving cybersecurity KG integrating traditional data sources and further demonstrated two use cases in vulnerability assessment and dynamic intrusion alert interpretation. However, mapping properties into nodes as shown in [13] would cause the problem of booming graph density and decreasing the information gain of relationships.

An in-depth analysis of security incidents shows that more and more exploits, also known as malware or malicious software, tend to attack multiple vulnerabilities jointly. Yin et al. [36] formulated a graph-based vulnerability risk assessment task, the VCBD problem, and built a **Graph Neural Network (GNN)**-based link prediction framework to predict if two vulnerabilities will be co-exploited or not.

Table 1. Data Collection Details

Object	Source	Common Attribute		Unique Attributes
		CVE ID	CWE ID	
Vulnerability	NVD	✓		description, vulnerability published date, available, reference links, CVSS V2 metrics, and CVSS V3 metrics
	CVE Details	✓	✓	gained access, vulnerability type, affected product(name, version, type, and vendor)
Weakness	CWE		✓	CWE name, description, CWE view, extended description, weakness abstraction, and status
Exploit	EDB	✓		exploit ID, exploit type, platform, exploit published date, and exploit author(name)

To summarize, although many research works have introduced graph approaches to vulnerability assessment, they all ignored the important exploit record data. Another common problem is they did not publish their original data and codes for direct reuse.

3 Software VulKG Construction

3.1 Data Collection, Analyzing, and Prepossessing

VulKD adopts the most commonly used databases as the data source, i.e., NVD, CVE Details, CWE database, and EDB. Table 1 shows the raw data we collected from existing relational databases.

(1) NVD. NVD is a comprehensive cybersecurity database managed by the National Institute of Standards and Technology that catalogs and indexes known vulnerabilities in software and hardware. Among all the attributes, CVE ID, also known as CVE Identifier or CVE name, is a unique, common identifier for publicly known cybersecurity vulnerabilities assigned by a CVE Numbering Authority, and can be used for identification and cross-source information integration. The CVSS is short for Common Vulnerability Scoring System (CVSS) scores, which provide a standardized way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The description is a detailed explanation of the vulnerability, including how it can be exploited and its potential impact on affected systems.
(2) CVE Details. The CVE Details database is an online public resource that provides detailed information about security vulnerabilities, which offers additional information such as gained access that facilitates a deeper understanding of vulnerabilities, their implications, and the scope of their impact.
(3) CWE. The CWE is a community-developed list of common software and hardware weakness types, and is designed to serve as a common language for describing software security weaknesses in architecture, design, or code. This database offers some unique information, such as weakness abstraction, that can be used to evaluate system risks based on semantic analysis methods. Especially, the CWE ID can serve as a linkage for its information being integrated with other databases.
(4) EDB. The EDB stores and manages concept-of-proof exploit records. Among all the attributes, exploit type categorizes the exploit based on its nature or the approach it uses, such as remote, local, and web applications, which helps users identify the context in which an exploit can be used. The CVE ID helps to build a link with other related databases such as NVD.

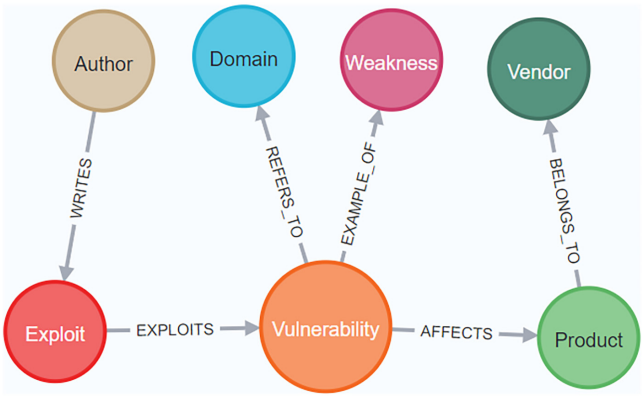


Fig. 1. The KG model of VulKG.

The details of the collected information can be found on the official websites of the aforementioned databases and our provided source data on GitHub.

3.2 KG Schema Design and Modeling

To design an efficient and flexible KG schema model for data from multiple sources, one needs to identify the entities and relationships along with their properties and labels from the collected raw data. According to the guidelines given by Neo4j,[8] this is an iterative process consisting of multiple steps: (1) understanding the application domain, including creating high-level sample data and defining specific questions for the application; (2) identifying entities, including entity properties; (3) identifying relationships including directions and relationship properties; (4) testing the questions against the model defined in steps (2) and (3); (5) testing the scalability; and (6) improving and refactoring the model.

VulKG is mainly designed for vulnerability risk assessment tasks. Based on the collected raw data, we first identify the objects shown in Table 1 as three entities, labeled as Vulnerability, Weakness, and Exploit. Then, we identified another four influential entities: Product, Vendor, Author and Domain. The relationships between entities are described as the KG model illustrated in Figure 1. When describing entities, relationships, and graph operations, we follow the Neo4j-recommended Cypher naming and coding standards,[9] where node labels are in CamelCase, property keys, variables, parameters, aliases, and functions are in camelCase, and relationship types are in upper-case.

The full list of entities and their properties are shown in Table 2. The third column is the UNIQUE constraint (equivalent to the primary key in a relational database) to ensure a specific node can be identified.

The relationships of VulKG are shown in Table 3. The second and third columns list the labels of head and tail entities. The fourth column shows the AFFECTS relationship has a 'version' property to describe the affected product version.

3.3 Deployment

We deploy VulKG on the Neo4j graph database platform. After deployment, the statistics of VulKG are shown in Table 4.

[8] https://neo4j.com/developer/guide-data-modeling/
[9] https://neo4j.com/docs/cypher-manual/current/syntax/naming/

Table 2. Entities of VulKG

Entity label	Property	Constraint
Vulnerability	CVE ID, description, published date, number of reference, CVSS v2 metrics, and CVSS v3 metrics	CVE ID
Exploit	exploit ID, exploit type, platform, and exploit published date	exploit ID
Weakness	CWE ID, description, CWE name, extended description, weakness abstraction, CWE view, status	CWE ID
Product	product name, product type	product name
Vendor	vendor name	vendor name
Author	author name	author name
Domain	domain name	domain name

Table 3. Relationships of VulKG

Type	Head	Tail	Property
EXPLOITS	Exploit	Vulnerability	none
AFFECTS	Vulnerability	Product	version
BELONGS_TO	Product	Vendor	none
EXAMPLE_OF	Vulnerability	Weakness	none
WRITES	Author	Exploit	none
REFERS_TO	Vulnerability	Domain	none

Table 4. Statistics of VulKG

Entity Label	Count	Relationship Type	Count
Vulnerability	148,609	EXPLOITS	28,791
Exploit	43,743	AFFECTS	212,654
Weakness	926	BELONGS_TO	47,509
Product	44,269	EXAMPLE_OF	71,031
Vendor	20,347	WRITES	43,743
Author	9,204	REFERS_TO	429,728
Domain	9,578		

Neo4j adopts Cypher as the declarative query language to create, read, update, or delete nodes and edges in a graph. Neo4j also provides interactive visual interfaces, like Neo4j Browser[10] and Neo4j Bloom,[11] to visualize the Cypher operation results directly. For example, when a vulnerability, "CVE-2018-19585," is disclosed, we can use the following Cypher codes to find out affected products and vendors, the existing exploits, and vulnerabilities that might be co-exploited with it.

[10] https://neo4j.com/docs/browser-manual/current/
[11] https://neo4j.com/product/bloom/

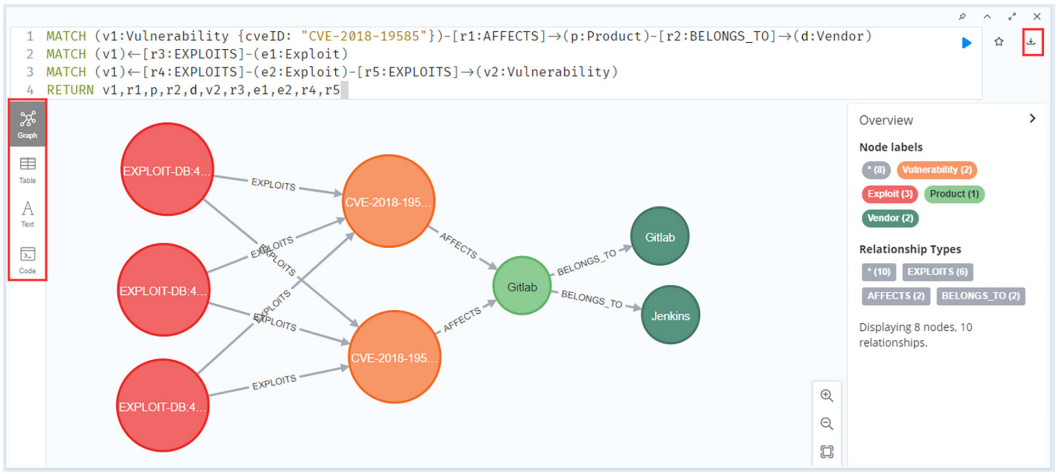


Fig. 2. Visualized result of the reasoning example.

Cypher:
MATCH (v1: Vulnerability cveID:"CVE-2018-19585")
-[r1:AFFECTS]-> (p:Product)
-[r2: BELONGS_TO]-> (d:Vendor)
MATCH (v1) <-[r3:EXPLOITS]-(e1:Exploit)
MATCH (v1) <-[r4:EXPLOITS]-(e2:Exploit)
-[r5:EXPLOITS]-> (v2:Vulnerability)
RETURN v1, r1, p, r2, d, v2, r3, e1, e2, r4, r5

The first MATCH clause uses the CVE ID to search for the exact vulnerability and return it as v1. Then, the affected products and vendors are searched by the AFFECTS and BELONGS_TO relationships and returned as p and d. The existing exploits, e1, are searched by the second MATCH Clause. The third MATCH clause can find the co-exploitation behavior, and the involved exploits and vulnerabilities are denoted as e2 and v2. Figure 2 shows the Cypher query result returned by Neo4j Browser. The affected product is Gitlab, which has two vendors, Gitlab and Jenkins. CVE-2018-19585 has three concept-of-proof exploits, i.e., EXPLOIT-DB:49257, EXPLOIT-DB:49263, and EXPLOIT-DB:49334. CVE-2018-19571 can be co-exploited with CVE-2018-19585 by all of these exploits. The results can also be presented in a table or plain text or exported as CSV, JSON, or PNG files.

3.4 Graph Statistics Compared with Similar Works

In order to show one of our major contributions, in Table 5, we summarized five previous works that are most similar to our work and have specified the statistics of their KG. As we can see from the table, the majority of the works did not publish their code and collected data, and the works in the first four rows failed to include the important EDB. Though the data source of [36] is very similar to our work, it mainly focused on developing a high-performance solution for co-exploit detection tasks instead of offering a KG for the community to reuse. Besides, the choice of entity and relation and the scale of the graph are also inferior to our work.

Table 5. Statistics of the Final VulKG Compared with Similar Works

Paper	Data Source						Graph Statistics				Open Source
	NVD	CVE	CWE	EDB	CPE	CAP -EC	Entity Types	Entity Count	Relation Types	Relation Count	
[26]	✓	✓	✓		✓		13 types	274,187	not given	566,279	No
[38]	✓	✓	✓			✓	3 types	5,416	9 types	10,502	No
[31]		✓	✓			✓	3 types	4,167	8 types	8,067	Yes
[29]		✓	✓			✓	3 types	4,144	12 types	16,746	No
[36]	✓	✓	✓	✓			4 types	256,971	5 types	288,956	No
Our	✓	✓	✓	✓			7 types	276,676	6 types	833,456	Yes

4 Use Case: Vulnerability Co-Exploitation Behavior Discovery

To demonstrate the process of applying VulKG to vulnerability assessment tasks and analyze the effectiveness of graph structural information, we provide a use case applying VulKG to solve the VCBD problem, which was first formulated as a link prediction problem in [36].

4.1 Problem Formulation

Let an isomorphic graph be denoted as $\mathcal{G} = \{\mathcal{V}, A, X\}$, where $\mathcal{V}$ is the vertex set, A is the adjacency matrix indicating if an edge exists or not between nodes u and $v \in \mathcal{V}$, and $X \in \mathbb{R}^{n \times |\mathcal{V}|}$ is the feature matrix extracted from nodes' properties, where n is the feature dimension and $|\mathcal{V}|$ is the total number of nodes in $\mathcal{V}$.

We formulate the link prediction problem on graph $\mathcal{G}$ as an edge-level binary classification problem. A typical data sample for link prediction between any two nodes u and v in $\mathcal{V}$ can be denoted as $\{x_{uv}, y_{uv}\}$ and x_{uv} can be calculated as Equation (1),

$$x_{uv} = f_a(x_u, x_v), (u, v \in \mathcal{V}), \tag{1}$$

where f_a is the aggregation function to integrate features of nodes u and v, such as a concatenate function. The ground truth, y_{uv}, can be labeled by Equation (2)

$$y_{uv} = \begin{cases} 1 & \text{if } A_{uv} = 1 \\ 0 & \text{if } A_{uv} = 0 \end{cases}, (u, v \in \mathcal{V}). \tag{2}$$

The predicted results, $\widehat{y}_{uv}$, can be calculated as Equation (3),

$$\widehat{y}_{uv} = f_c(x_{uv}, \Theta_c), (u, v \in \mathcal{V}), \tag{3}$$

where $f_c(\cdot)$ is the mapping function from input x_{uv} to the output of a binary classifier, and Θ_c is the trainable parameters of the classifier, which can be solved by minimizing the loss function defined on the link prediction dataset D derived from $\mathcal{G}$, as shown in Equation (4),

$$\min_{\Theta_c} \mathcal{L} = f_L(f_l(y_{uv}, \widehat{y}_{uv}), D), \tag{4}$$

where $f_l(\cdot)$ is the cost function for a single data sample and f_L is the loss function over the whole dataset D.

4.2 Experiment

4.2.1 Tailored Subgraph Construction. Instead of being an isomorphic graph, VulKG is a heterogeneous graph with multiple entity labels and relationship types. Since VulKG is deployed on the Neo4j graph database platform, it is easy to refactor and tailor VulKG to a desired isomorphic subgraph. This subsection provides a use case for refactoring and tailoring VulKG for the VCBD task.

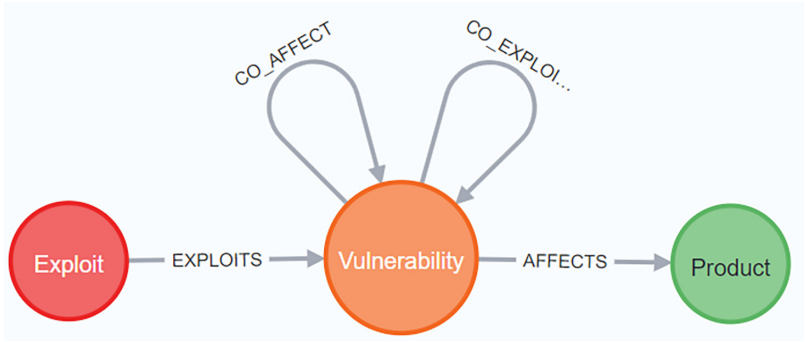


Fig. 3. The schema of the tailored subgraph for VCBD.

We first refactor VulKG by adding an additional CO_EXPLOITATION relationship between vulnerabilities v1 and v2 when the path (v1:Vulnerability)<-[:EXPLOITS]-(:Exploit)-[:EXPLOITS]->(v2:Vulnerability) exists. Then, we further refactor VulKG by adding a CO_AFFECT relationship between vulnerabilities v1 and v2 if a path (v1:Vulnerability)-[:AFFECTS]->(:Product)<-[:AFFECTS]-(v2:Vulnerability) exists. As a result, we can leverage the graph connections between vulnerabilities and their affected products to enhance the performance of the VCBD problem. Finally, the schema of the tailored subgraph is shown in Figure 3.

4.2.2 Dataset Construction. When constructing training and test datasets from a graph, random splitting of nodes and relationships can easily lead to data leakage, where nodes and relationships in the training set might connect with those in the test set. To prevent this, we split the nodes in the tailored subgraph based on vulnerability publication dates. This temporal split ensures that all nodes and relationships in the training set precede those in the test set, thereby preserving the chronological order of the data. After this temporal split, we extract the training and test subgraphs separately, ensuring that there are no connections between the two sets and thus preventing any information leakage from future states into past states.

In this use case, we start with using an isomorphic graph consisting of Vulnerability entities and CO_EXPLOITATION relationships to generate labeled training and test datasets. To balance the datasets, we retain all existing CO_EXPLOITATION links as positive samples and randomly down-sample an equal number of negative links. Specifically, the graph contains a total of 6,880 CO_EXPLOITATION links. Of these, 5,349 links occurring before 2015 are designated as positive links for the training set, while the remaining 1,531 links occurring after 2015 are assigned to the test set. Consequently, the training set comprises 10,698 samples, and the test set comprises 3,062 samples. Links are labeled in accordance with Equation (2).

Then, using the same Vulnerability entities in the training and test datasets, we extract isomorphic subgraphs consisting of Vulnerability entities and CO_AFFECT relationships to extract topological features of Vulnerability entities. We group the extracted features for each Vulnerability node into non-topological and topological features to facilitate subsequent analysis and discussion. Specifically, the non-topological features are extracted from the properties of Vulnerability entities listed in Table 2. We adopt a pre-trained bidirectional encoder representations from transformers model to embed the description and then decrease the dimension of embedding to 20 using a principal component analysis algorithm following [34]. Numerical properties remain unchanged. Boolean properties are converted to integer features, and categorical properties are represented in

Table 6. Feature Extraction Details

Group	Sub-Group	Dimension	Method
Non-topological	cvss	34	general preprocessing
	desc	40	pre-trained BERT model
Topological	centra	8	page rank, article rank, degree centrality, and harmonic centrality
	commun	8	Louvain, label propagation, weakly connected components, and modularity optimization
	fastRP	40	fast random projection
	node2vec	40	Node2Vec

Table 7. Classifiers and Parameter Settings (scikit-learn==1.1.1)

Name	Description	Parameter Setting
KNN	K Nearest Neighbors	$k = 5$
MLP	Multi-layer Perceptron	alpha = 1e-5, hidden_layer_sizes = 64, activation = 'logistic,' learning_rate = 'adaptive,' early_stopping = True
LR	Logistic Regression	random_state = 0, solver = 'liblinear,' max_iter = 2,000
DT	Decision Tree	max_depth = 10
RF	Random Forest	max_depth = 10, n_estimators = 9, max_features = 1
ABC	AdaBoost	base_estimator = DT, n_estimators = 9
GNB	Gaussian Naive Bayes	Default

ordinal encoding format. Topological features are extracted from the aforementioned CO_AFFECT isomorphic subgraph following the built-in functions of the Neo4j Graph Data Science library.[12]

The detailed feature extraction methods and the dimensions of extracted features are summarized in Table 6. The non-topological feature group (Nontopo) contains two sub-groups, i.e., CVSS metrics and description, denoted as cvss and desc, respectively, following the practice in [33, 34]. The Topological feature group (Topological) has four sub-groups, i.e., the centrality features (centra), the community detection features (commun), the fast random projection node embedding features (fastRP), and the Node2Vec node embedding features (node2vec).

4.2.3 Experiment Design. To compare the performance of features extracted from Non-topological and Topological feature groups and discuss the effectiveness of structural information, we traverse different feature combinations across groups and compare multiple classifiers on the constructed VCBD dataset. The classifiers are chosen from the scikit-learn[13] package, and Table 7 shows their parameter settings.

It is worth mentioning that sophisticated and fine-tuned classifiers always produce a higher performance on any of those feature groups. As such, the goal of our experiment is to investigate the relative representing power of different feature groups rather than competing to generate the best classification performance.

[12] https://neo4j.com/docs/graph-data-science/current/algorithms/
[13] https://scikit-learn.org/stable/

Table 8. Cross-Group Best Performance Comparison

Classifier	Feature	Acc(%)	Pre(%)	Rec(%)	F1(%)
MLP	**Topo_centra**	**85.73**	**86.3**	**85.73**	**85.67**
	Nontopo_desc	69.69	70.63	69.69	69.35
RF	**Topo_commun**	**84.75**	**85.37**	**84.75**	**84.68**
	Nontopo_cvss	61.23	65.50	61.23	58.37
ABC	Topo_commun	84.49	84.53	84.49	84.48
	Nontopo_cvss	62.44	64.29	62.44	61.19
DT	Topo_commun	82.33	82.52	82.33	82.31
	Nontopo_cvss	67.44	67.51	67.44	67.41
KNN	Topo_centra	81.74	81.75	81.74	81.74
	Nontopo_desc	70.77	70.80	70.77	70.76
LR	Topo_centra	58.36	61.16	58.36	55.57
	Nontopo_desc	53.14	53.20	53.14	52.90
GNB	Topo_fastRP	56.14	57.25	56.14	54.39
	Nontopo_desc	55.26	57.49	55.26	51.65

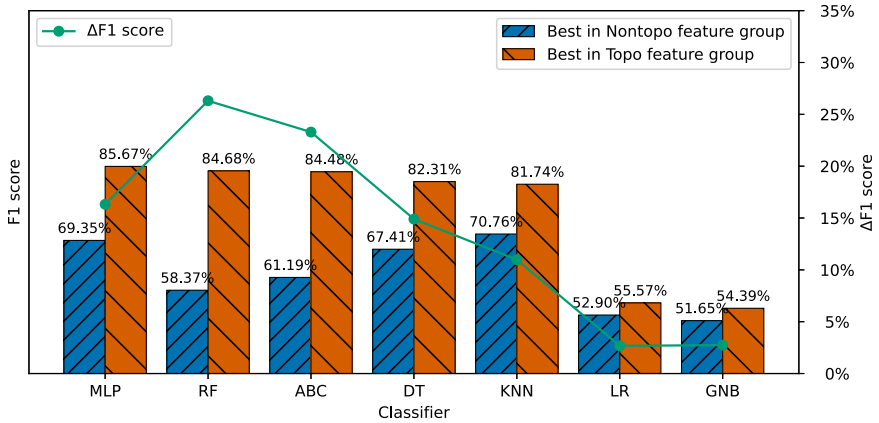


Fig. 4. Best achieved F1 score comparison across feature groups.

4.3 Results and Analysis

4.3.1 Cross-Group Feature Power Comparison. Table 8 reports the best sub-group performance comparison on the Topological and Non-topological feature groups for each tested classifier. Though the best-performed sub-group feature may differ according to the chosen classifier, features from the Topological group always produce a more satisfying result than those from the Non-topological group. The consistent results show the effectiveness of structural information in addressing the connection-oriented VCBD problem.

Figure 4 visualizes the best achieved F1 score comparison across different feature groups. The ΔF1 score line in blue clearly illustrates the performance gain coming along with structural information. The dominant representing power of topological features shown in Figure 4 enables us to argue that our constructed VulKG could serve a positive role in certain research directions for vulnerability assessment.

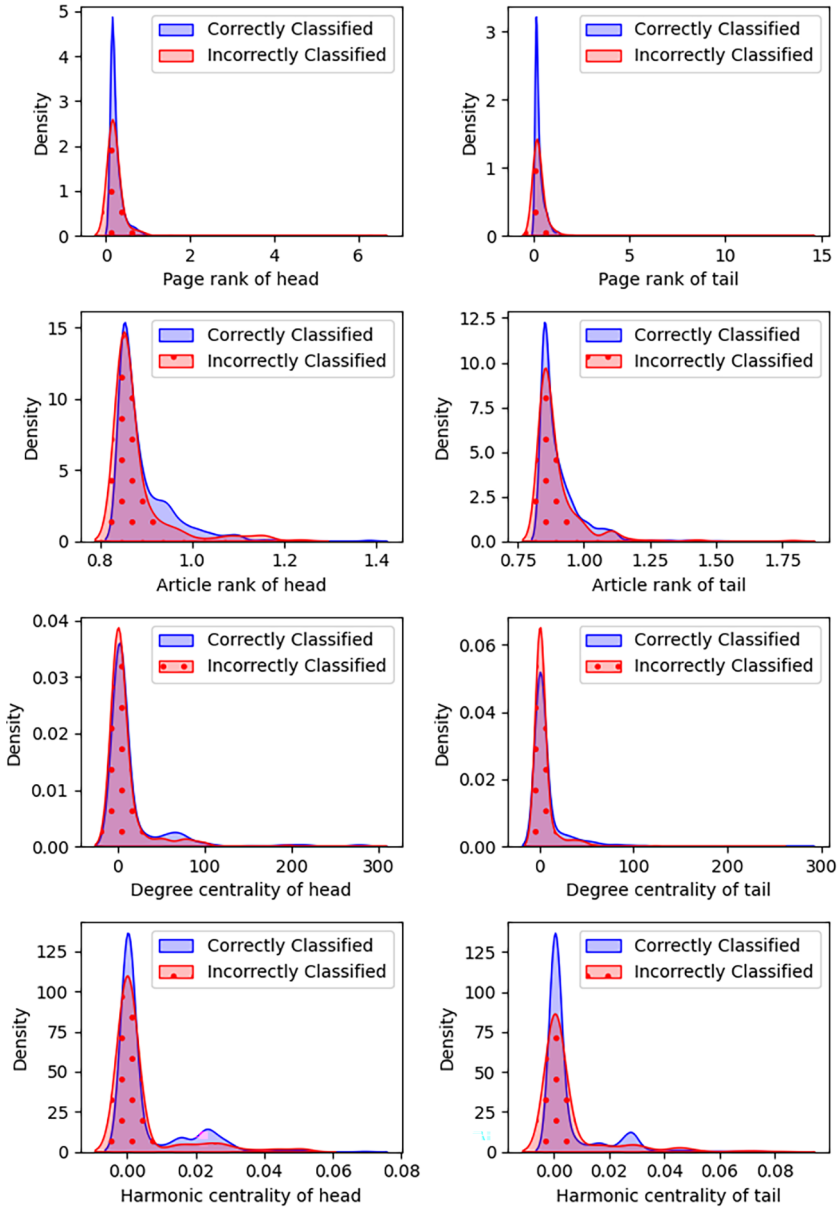


Fig. 5. Distribution of Topo_centra features.

To provide a more in-depth analysis of the cases that were properly classified and those that were not, we further examined the best results reported in Table 8. These results were achieved using an **Multi-Layer Perceptron (MLP)** classifier, with parameter settings detailed in Table 7. The feature set, Topo_centra, includes eight columns derived from four centrality algorithms (page rank, article rank, degree centrality, and harmonic centrality) applied to both the head and tail nodes of a link, as outlined in Table 6. We have drawn distribution plots for each of these eight columns, distinguishing between correctly classified and incorrectly classified samples. The results are presented in Figure 5.

Table 9. Feature Integration Effect on Top-Two Performed Classifiers

Classifier	Feature	Acc(%)	Pre(%)	Rec(%)	F1(%)	AUC(%)
MLP	Topo_centra	85.73	86.31	85.73	85.67	91.20
	Nontopo_desc + Topo_centra	81.91	82.93	81.91	81.77	91.00
	Nontopo_desc	69.69	70.63	69.69	69.35	77.30
RF	Topo_commun	84.75	85.37	84.75	84.68	89.80
	Nontopo_cvss + Topo_commun	61.07	61.98	61.07	60.32	68.40
	Nontopo_cvss	61.23	65.50	61.23	58.37	69.40

The density plots for both sets of samples exhibit similar shapes and are leptokurtic, characterized by tall and narrow peaks with heavy tails. However, there are notable differences:

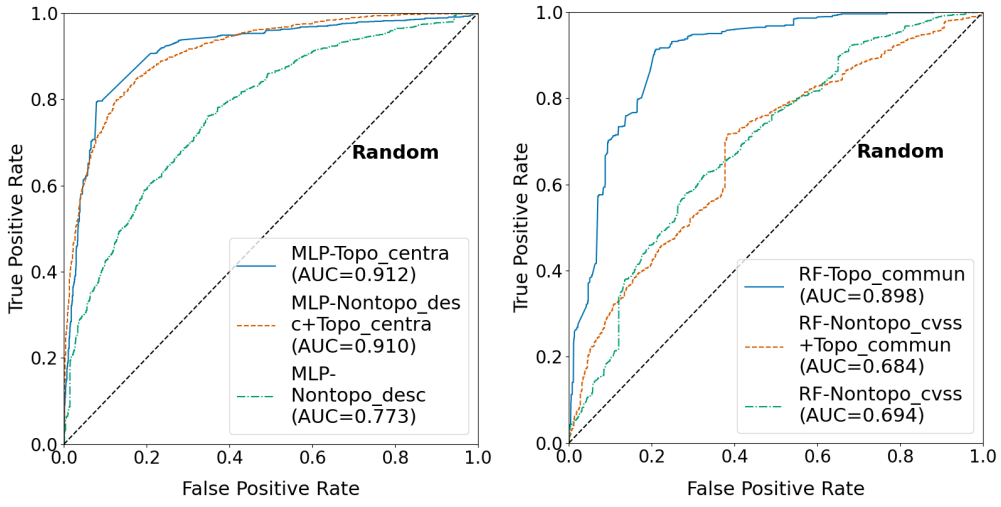
—Correctly Classified Samples: Most of the density plots for these features are taller and narrower, indicating that the distributions have smaller spreads and less variability. This suggests that the features of correctly classified samples tend to be more predictable and consistent, with values clustering closely around the mean.
—Incorrectly Classified Samples: The density plots for these features are generally flatter and wider, indicating higher variability and more diverse values. This greater spread suggests that it is more challenging to make correct classifications, as the model must consider a broader range of values and a higher possibility of extreme values.

4.3.2 Feature Integration Effect. In this part, we pick out the top two classifiers in the previous section, MLP and **Random Forest (RF)**, and combine the best-performed features in each group for those two classifiers, respectively. Table 9 shows the performance comparison of the best-performed feature from both Topological and Non-topological groups and their integration. The results in row 2 echo with those in row 5 that a simple combination of the best features in Topological and Non-topological groups does not necessarily guarantee performance improvement.

Figure 6 shows the corresponding ROC performance comparison on feature integration and the individual feature. As shown in Figure 6(a), the single Topo_centra feature achieves the best performance for the MLP classifier, which also is the best performance for this task. However, when combining the best Non-topological feature, Nontopo_des, with Topo_centra, classification performance suffers a slight degeneration from AUC = 0.912 to AUC = 0.910, which is not unique to the MLP classifier. In Figure 6(b), the same degeneration situation happens for the RF classifier, and in an even more severe way.

Our guess is that the characteristics of Non-topological features extracted from node properties, such as CVSS metrics, may not behave in the same direction as the characteristics of Topological features extracted from graph connections. When combining them in a simple way, such as concatenation, their effects are, to some extent, weakened by each other. Actually, this kind of performance downgrade also exists even when adopting GNN-based graph embedding methods, which favor the advanced infusion of node property and graph connection. For example, [36] proposed a modality-aware graph convolutional networks that achieved 81.34% of F1 score on the same VCBD problem, though better than GCN (78.96%) [14], GraphSAGE (79.62%) [9], EdgeGCN (77.96%) [30], and GINGCN (80.55%) [32], it is still inferior to the F1 score achieved by the Topo_centra feature group here at 85.73%.

Therefore, to maximize the performance of a specific vulnerability risk assessment task, we suggest always trying different graph feature extraction algorithms and investigating



(a) MLP classifier (b) RF classifier

Fig. 6. ROC comparison on feature integration.

combinations of topological and non-topological features when applying the graph information in VulKG.

5 Conclusion

This article introduces a compact VulKG, VulKG, for software vulnerability risk assessment and deploys it on the Neo4j graph database platform. Especially, we provide a use case on vulnerability co-exploitation behavior discovery task to demonstrate the flexibility of refactoring and extracting subgraphs from VulKG for self-defined problems. With VulKG, several benefits are expected for the community: first, researchers in this field could be largely relieved from data collecting and processing, focusing more on developing efficient detection models. Second, with this scale of data and the flexibility of using it, more innovative research problems could be studied. For example, VulKG can be used to develop VEP and VETP algorithms by formulating them as node-level classification problems. Third, as a prototype of an integrated vulnerability database, VulKG can be extended to customized risk assessment tools for organizations, enabling agile patch management.

Nevertheless, our work also faces some limitations in data collection and system dynamics. On one hand, the effectiveness of VulKG is heavily dependent on the completeness and accuracy of the integrated databases. Inaccuracies or omissions in the data can lead to incorrect assessments and potentially overlook critical vulnerabilities. On the other hand, the cybersecurity landscape is constantly evolving. With new vulnerabilities and exploits emerging at a rapid pace, Keeping VulKG up-to-date requires continuous monitoring and integration of new data sources, which can be a resource-intensive process for the current static version of VulKG. Therefore, we will focus on the automatic updating of the KG in future work while also looking forward to other detection models inspired by VulKG.

References

[1] Ahmed AlEroud and George Karabatis. 2012. A contextual anomaly detection approach to discover zero-day attacks. In *Proceedings of the 2012 International Conference on Cyber Security*. IEEE, 40–45.

[2] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2010. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 105–114.

[3] Haipeng Chen, Jing Liu, Rui Liu, Noseong Park, and V. S. Subrahmanian. 2019. VEST: A system for vulnerability exploit scoring & timing. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*. 6503–6505.

[4] Haipeng Chen, Rui Liu, Noseong Park, and V. S. Subrahmanian. 2019. Using twitter to predict when vulnerabilities will be exploited. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3143–3152.

[5] Dongdong Du, Xingzhang Ren, Yupeng Wu, Jien Chen, Wei Ye, Jinan Sun, Xiangyu Xi, Qing Gao, and Shikun Zhang. 2018. Refining traceability links between vulnerability and software component in a vulnerability knowledge graph. In *Proceedings of the International Conference on Web Engineering*. Springer, 33–49.

[6] Michel Edkrantz and Alan Said. 2015. Predicting cyber vulnerability exploits with machine learning. In *Proceedings of the 13th Scandinavian Conference on Artificial Intelligence (SCAI 15)*. 48–57.

[7] Seyed M. Ghaffarian and Hamid R. Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys (CSUR)* 50, 4 (2017), 1–36.

[8] Seyed M. Ghaffarian and Hamid R. Shahriari. 2021. Neural software vulnerability analysis using rich intermediate graph representations of programs. *Information Sciences* 553 (2021), 189–207.

[9] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[10] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME '17)*. IEEE, 125–136.

[11] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Michael Roytman, and Idris Adjerid. 2019. Exploit prediction scoring system (EPSS), Digital Threats: Research and Practice. 2: 1 - 17. Retrieved from https://api.semanticscholar.org/CorpusID:199577534

[12] Yan Jia, Yulu Qi, Huaijun Shang, Rong Jiang, and Aiping Li. 2018. A practical approach to constructing a knowledge graph for cybersecurity. *Engineering* 4, 1 (2018), 53–60. DOI: https://doi.org/10.1016/j.eng.2018.01.004

[13] Elmar Kiesling, Andreas Ekelhart, Kabul Kurniawan, and Fajar Ekaputra. 2019. The SEPSES knowledge graph: An integrated resource for cybersecurity. In *Proceedings of the International Semantic Web Conference*. Springer, 198–214.

[14] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. Retrieved from https://api.semanticscholar.org/CorpusID:3144218

[15] Xiangjie Kong, Menglin Li, Jianxin Li, Kaiqi Tian, Xiping Hu, and Feng Xia. 2019. CoPFun: An urban co-occurrence pattern mining scheme based on regional function discovery. *World Wide Web* 22 (2019), 1029–1054.

[16] Xiangjie Kong, Yajie Shi, Shuo Yu, Jiaying Liu, and Feng Xia. 2019. Academic social networks: Modeling, analysis, mining and applications. *Journal of Network and Computer Applications* 132 (2019), 86–103.

[17] Xiangjie Kong, Ximeng Song, Feng Xia, Haochen Guo, Jinzhong Wang, and Amr Tolba. 2018. LoTAD: Long-term traffic anomaly detection based on crowdsourced bus trajectory data. *World Wide Web* 21 (2018), 825–847.

[18] Triet H. M. Le, Huaming Chen, and M. Ali Babar. 2021. A survey on data-driven software vulnerability assessment and prioritization. *ACM Computing Surveys (CSUR) 55,* (2021), 1-39.

[19] Fan Liu, Xingshe Zhou, Jinli Cao, Zhu Wang, Hua Wang, and Yanchun Zhang. 2019. Arrhythmias classification by integrating stacked bidirectional LSTM and two-dimensional CNN. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 136–149.

[20] Francesco Lomio, Emanuele Iannone, Andrea De Lucia, Fabio Palomba, and Valentina Lenarduzzi. 2022. Just-in-time software vulnerability detection: Are we there yet? *Journal of Systems and Software* 188, (2022), 111283.

[21] Jie Lu, Zheng Yan, Jialin Han, and Guangquan Zhang. 2019. Data-driven decision-making (D 3 M): Framework, methodology, and directions. *IEEE Transactions on Emerging Topics in Computational Intelligence* 3, 4 (2019), 286–296.

[22] Steven Noel, Eric Harley, Kam Him Tam, Michael Limiero, and Matthew Share. 2016. CyGraph: Graph-based analytics and visualization for cybersecurity. In: Venkat N. Gudivada, Vijay V. Raghavan, Venu Govindaraju, C.R. Rao (Eds.), *Handbook of Statistics*, Vol. 35. Elsevier, 117–167.

[23] Shengzhi Qin and K. P. Chow. 2019. Automatic analysis and reasoning based on vulnerability knowledge graph. In *Proceedings of the Cyberspace Data and Intelligence, and Cyber-Living, Syndrome, and Health*. Springer, 3–19.

[24] Ernesto R. Russo, Andrea Di Sorbo, Corrado A. Visaggio, and Gerardo Canfora. 2019. Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. *Journal of Systems and Software* 156 (2019), 84–99.

[25] Carl Sabottke, Octavian Suciu, and Tudor Dumitraş. 2015. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *Proceedings of the 24th* {USENIX{USENIX} Security 15). *1041–1056.*

[26] Zhenpeng Shi, Nikolay Matyunin, Kalman Graffi, and David Starobinski. 2022. Uncovering product vulnerabilities with threat knowledge graphs. In *Proceedings of the IEEE Secure Development Conference (SecDev '22).* IEEE, 84–90.

[27] Yizhen Sun, Dandan Lin, Hong Song, Minjia Yan, and Linjing Cao. 2020. A method to construct vulnerability knowledge graph based on heterogeneous data. In *Proceedings of the 16th International Conference on Mobility, Sensing and Networking (MSN '20).* IEEE, 740–745.

[28] Nazgol Tavabi, Palash Goyal, Mohammed Almukaynizi, Paulo Shakarian, and Kristina Lerman. 2018. Darkembed: Exploit prediction with neural language models. In *Proceedings of the 32 AAAI Conference on Artificial Intelligence*. 7849–7854.

[29] Yan Wang, Xiaowei Hou, Xiu Ma, and Qiujian Lv. 2022. A software security entity relationships prediction framework based on knowledge graph embedding using sentence-bert. In *Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 501–513.

[30] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.

[31] Hongbo Xiao, Zhenchang Xing, Xiaohong Li, and Hao Guo. 2019. Embedding and predicting software security entity relationships: A knowledge graph based approach. In *Proceedings of the 26th International Conference on Neural Information Processing (ICONIP '19)*, Part III 26. Springer, 50–63.

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? Retrieved from https://api.semanticscholar.org/CorpusID:52895589

[33] Jiao Yin, MingJian Tang, Jinli Cao, and Hua Wang. 2020. Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description. *Knowledge-Based Systems* 210 (2020), 106529. DOI: https://doi.org/10.1016/j.knosys.2020.106529

[34] Jiao Yin, MingJian Tang, Jinli Cao, Hua Wang, and Mingshan You. 2022. A real-time dynamic concept adaptive learning algorithm for exploitability prediction. *Neurocomputing* 472 (2022), 252–265.

[35] Jiao Yin, MingJian Tang, Jinli Cao, Hua Wang, Mingshan You, and Yongzheng Lin. 2020. Adaptive online learning for vulnerability exploitation time prediction. In *Proceedings of the Web Information Systems Engineering (WISE '20)*. Springer, 252–266.

[36] Jiao Yin, MingJian Tang, Jinli Cao, Mingshan You, Hua Wang, and Mamoun Alazab. 2022. Knowledge-Driven Cybersecurity intelligence: Software Vulnerability Co-exploitation Behaviour Discovery. *IEEE Transactions on Industrial Informatics* 19, 4 (2022), 5593–5601.

[37] Mingshan You, Jiao Yin, Hua Wang, Jinli Cao, Kate Wang, Yuan Miao, and Elisa Bertino. 2022. A knowledge graph empowered online learning framework for access control decision-making. *World Wide Web* 26, (2022), 1–22.

[38] Liu Yuan, Yude Bai, Zhenchang Xing, Sen Chen, Xiaohong Li, and Zhidong Deng. 2021. Predicting entity relations across different security databases by using graph attention network. In *Proceedings of the IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC '21).* IEEE, 834–843.

[39] Shichao Zhu, Shirui Pan, Chuan Zhou, Jia Wu, Yanan Cao, and Bin Wang. 2020. Graph geometry interaction learning. *Advances in Neural Information Processing Systems* 33 (2020), 7548–7558.

Received 16 March 2023; revised 27 May 2024; accepted 31 May 2024