International Journal of Information Security and Privacy
Volume 18 • Issue 1

VCGERG:
Vulnerability Classification With Graph Embedding Algorithm on Vulnerability Report Graphs

Yashu Liu, Beijing University of Civil Engineering and Architecture, China
https://orcid.org/0000-0002-7208-5360
Xiaoyi Zhao, Beijing University of Civil Engineering and Architecture, China*
Xiaohua Qiu, Beijing University of Civil Engineering and Architecture, China
Han-Bing Yan, National Computer Network Emergency Response Technical Team, China

ABSTRACT

Vulnerability can lead to data loss, privacy leakage and financial loss. Accurate detection and identification of vulnerabilities is essential to prevent information leakage and APT attacks. This paper explores the possibility of digging the valuable information in vulnerability reports deeply. We propose a new model, VCGERG, which products a graph using key information from vulnerability reports and embeds the graph into the vector space using a keywords-LINE graph embedding algorithm based on the attention of neighboring nodes. VCGERG model uses the OVR random forest algorithm to classify vulnerabilities. Our model can get the complicated local and global information of the graph in large-scale dataset and achieve better results. In order to verify the effectiveness of our model, it is evaluated on many experiments. Compared with other models, our method has a higher accuracy rate of 0.975.

KEYWORDS

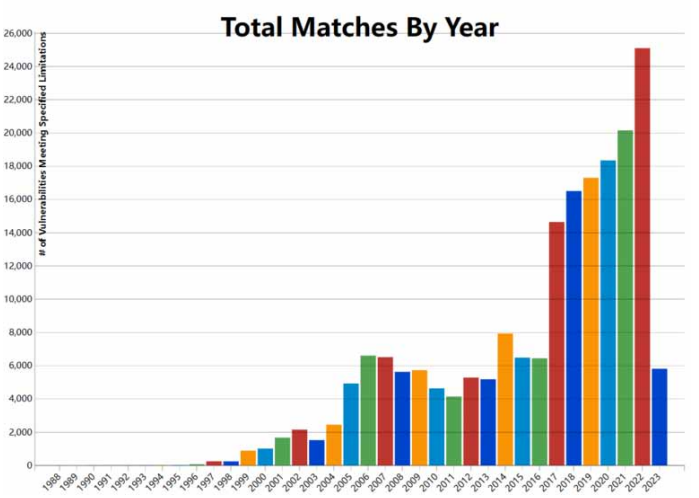
Graph Embedding, Graph Structure, Vulnerability Classification, Vulnerability Reports

Vulnerability is a flaw in the specific implementation of hardware, software, protocols, or system security policies. It is used by attackers to access or destroy the system without authorization. The National Vulnerability Database (NVD) (National Institute of Standards and Technology [NIST], 2023a) provides data on the subject as shown in Figure 1. The figure shows that the total number of vulnerabilities reached 25,102 in 2022, a significant increase of 24.5% compared with the previous year. Thus, the escalating severity of cybersecurity issues has prompted researchers to focus on identifying and classifying these vulnerabilities. However, the vast number and heterogeneity of vulnerability information pose a significant challenge to be able to detect and classify vulnerabilities quickly.

Certain institutions have now organized identified vulnerabilities. Common Weakness Enumeration (CWE) (2023) provides an extensive list of prevalent security weaknesses, where each

DOI: 10.4018/IJISP.342596
*Corresponding Author

Figure 1. Amount of Vulnerability Data Released by NVD



one corresponds to a different class of software bugs or defects. The latest CWE list, version 4.14, lists 1,362 different CWE-IDs and provides a comprehensive framework for identifying and classifying common security weaknesses. Researchers can facilitate systematic examination and analysis of the vulnerabilities using this framework. Attackers exploit vulnerabilities taking advantage of bugs or flaws in software systems. For example, microweber security vulnerability (CVE-2022-2353) (NIST, 2023b) appeared in July 2022. It is classified as a "cross-site request forgery" (CSRF) vulnerability (CWE-352), which allows an attacker to steal cross-site request forged tokens, access content from the same site, and redirect. This vulnerability poses risks such as data leakage and user privacy losses. Another notable example is the Apple iOS and macOS out-of-bounds write vulnerability (CVE-2022-32893) (NIST, 2023c), which appeared in August 2022. It is classified as an "out-of-bounds write" vulnerability (CWE-787).

In addition to CWE, vulnerability reports released by the NVD and security vendors have significant value in understanding and discovering vulnerability characteristics. The reports include summary information, the operating system and software type, threatening behavior, MD5, and other text information. The reports are unstructured and consist of text and tables. We can extract and reorganize the information from a vulnerability report. We can then describe vulnerability features from a more diverse perspective and construct a vulnerability feature library, which can be used to detect vulnerabilities automatically. In security and confidentiality applications, timely deep learning analysis and classification of data and information are of great significance (Aljarf et al., 2023; Altalhi & Gutub, 2021; Sufi et al., 2023). Using machine learning and deep learning methods further improves the accuracy of the classification results compared with traditional methods (Aljarf et al., 2023; Gutub et al., 2023; Hemalatha et al., 2023; Roy et al., 2023). This paper is dedicated to obtaining and analyzing the information in vulnerability reports and classifying the vulnerabilities.

We propose a graph structure to represent heterogeneous information in vulnerability reports, incorporate the LINE algorithm based on the attention mechanism of neighboring nodes to characterize vulnerabilities, and use the one VS rest (OVR) random forest model to identify the vulnerabilities.

Our three main contributions with this research are as follows:

1. Vulnerability report graph representation method. We propose an unstructured and heterogeneous information representation method using a graph. It breaks the limitation of the traditional approach

that only analyzes certain vulnerability features, such as attack behaviors, which enables us to build connections between multilevel and multiangle vulnerability features through the graph.

2. Understanding and extracting features of vulnerability report graphs. The vulnerability report graph is built by summary information, the operating system and software type, threatening behavior, and CWE-ID. To illustrate how to understand the graph and represent its features, we propose a new "vulnerability classification with a graph embedding algorithm on vulnerability report graphs" (VCGERG) model that combines the keywords-LINE graph embedding algorithm with the attention mechanism of neighboring nodes. Thus, from a large number of vulnerability reports, our approach lends to discover potential knowledge, patterns, and regularities that can help improve the analysis strategy and performance of vulnerability discovery.
3. Experiments and result analysis. We use the OVR random forest model to identify and classify vulnerabilities. Outlined in this paper, we designed many experiments on several datasets. The experimental results show that our method improves all evaluation metrics significantly, which verifies the effectiveness of our proposed model. Our method has high accuracy of 0.975.

In the next section, we detail several recent research efforts related to vulnerability studies. Then we describe the research methodology. We then offer a review of the process of validation by comparing ours with other methods to illustrate its accuracy and effectiveness. Finally, we summarize our work and discuss possible future research directions in the last section.

RELATED WORKS

Sun et al. (2021) proposed a vulnerability detection model, VDSimilar, based on code similarity using BiLSTM and attention network integrated into the Siamese model to get the similarity between two vulnerability functions and the difference between the vulnerability function and the patch function. By comparing the tested program to known vulnerability codes to discriminate whether the code is vulnerable, Hu et al. (2023) extracted slices of C/C++ source code and implemented an efficient and accurate vulnerability detection and interpretation method using a graph neural network. Zou et al. (2019) proposed a multiclass vulnerability detection system. They introduced the concept of code attention, using local features to detect vulnerability types. They completed multiclass vulnerability detection by considering program control dependencies during program slice construction. Wartschinski et al. (2022) constructed the Vudenc vulnerability detection model to implement Python code detection. Python codes are trained by a word2vec model and represented as vectors. The LSTM (Long Short-Term Memory) network then classifies the sequence of vulnerable code tokens at a fine-grained level and highlights with different colors specific regions of the source code that may contain vulnerabilities.

Compared with source code, vulnerability reports can represent the characteristic information of vulnerability more intuitively and are released by authoritative security organizations and vendors. The content is reliable and trustworthy. Aljedaani et al. (2020) used the latent Dirichlet allocation (LDA) to classify security bug reports (SBRs) in the Chromium project. They found the potential topics in the SBR text and proved they were very close to vulnerability types. Alperin et al. (2020) used the LIME model to interpret vulnerability description and proposed the GenSim latent semantic indexing module to create a latent semantic analysis (LSA) for each category. Aota et al. (2020) vectorized the vulnerability information on the NVD with a bag-of-words model (BoW). They used the Boruta algorithm to select meaningful features (e.g., CWE-ID) and random forest (RF) for classification. Han et al. (2017) used only vulnerability descriptions and CVSS (2022) scores from the CVE database (CVE, 2017) to predict vulnerability severity. It used a skip-gram word vector model and a single-layer CNN for classification. Han et al. (2018) extracted the CWE-ID text description and expected consequences of the vulnerability report on CWE to construct a knowledge graph. Additionally, they

used a graph embedding algorithm to get a low-dimensional vector, effectively supporting various inference tasks for vulnerabilities.

Graph embedding algorithms aim to map nodes to low-dimensional vectors in a graph, preserving the structural relationships between nodes and allowing otherwise complex graph data to be represented more efficiently in vector space. DeepWalk (Perozzi et al., 2014) captures the structural information of a graph by randomly wandering through it and learning the low-dimensional representations using the skip-gram model. Node2vec (Grover & Leskovec, 2016) captures the structural information of a graph by introducing flexible stochastic wandering strategies, including breadth-first and depth-first, to balance local and global structures. LINE (Tang et al., 2015) is specifically designed to deal with large-scale datasets in the real world. It plays a vital role in representing high-dimensional graph structures in low-dimensional vector spaces using the local and global structure between nodes and nodes in the graph. LINE graph embedding algorithm optimizes the local structure vector of each node in the graph using first-order similarity. It optimizes the global structure vector using second-order similarity.

These studies, however, did not use the total content of the current report; they either focused on the vulnerability description part or other single attributes such as CWE-ID. Certain descriptions in the report, however, could more precisely find useful information. In addition, natural language may have limitations in analyzing vulnerability text information. To analyze the report more comprehensively and effectively, it is necessary to explore more dimensions of information.

VCGERG MODEL

The VCGERG model has three parts. To better illustrate the framework proposed in this paper, we first provide an overview of the framework before outlining the details. The following subsection describes the keywords extraction method and vulnerability report graph construction method. Then a review of the main principles of the LINE graph embedding model is presented, followed by a description of the neighbor nodes' attention mechanism proposed in this paper. An introduction of the OVR random forest multi-classification model concludes this section.

Traditional LINE algorithms are capable of comprehensively analyzing structured vulnerability report information. However, it cannot analyze unstructured vulnerability descriptions effectively. Vulnerability description is an overview of vulnerability, which is unstructured textual information provided by authoritative domain experts. Vulnerability description is an essential source of information for vulnerability report analysis.

While dealing with long text, the LINE algorithm cannot capture the fine-grained semantic relationships between all words or deal with long-distance semantic associations. The LINE algorithm may lose critical information. In this paper, we propose a novel keywords-LINE graph embedding model that integrates the neighbor node attention mechanism to process vulnerability description directly.

Figure 2 illustrates the framework of the proposed method, which depicts three closely connected modules: the vulnerability report graph construction module, the vulnerability report graph vectorized module, and the vulnerability classification module. The vulnerability report graph construction module is responsible for obtaining and processing the relevant information in the reports. Then, the LINE graph embedding algorithm, which contains the attention mechanism of neighbor nodes, vectorizes the vulnerability report graph node information and transforms the graph into a low-dimensional vector space. Finally, the vulnerability classification module employs the OVR random forest algorithm to classify vulnerabilities accurately.

Construction of Vulnerability Reporting Diagrams

Vulnerability reports contain structured and unstructured data. The structured portion provides intuitive information about the characteristics of the vulnerability such as attack path and attack complexity.

Figure 2. Structural Framework of the Vulnerability Report Analysis Classification

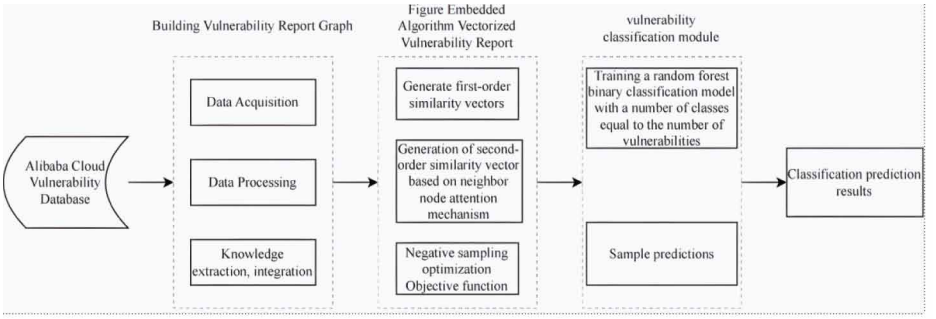


Figure 3. Small Portion of Structured and Unstructured Data

ATTACK PATH	ATTACK COMPLEXITY	Vulnerability description
local	difficulty	Prior to microweber/microweber v1.2.20, due to improper neutralization of input, an attacker can steal tokens to perform cross-site request forgery, fetch contents from same-site and redirect a user.

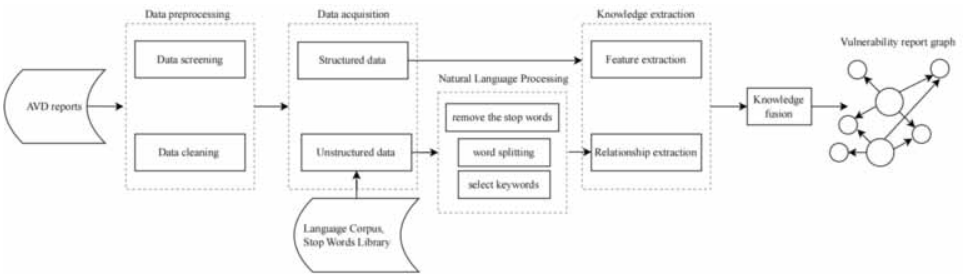
The unstructured vulnerability description text is often long, complex, and diverse. Figure 3 shows a small portion of structured and unstructured data from the intercepted Alibaba Cloud Vulnerability Database (AVD) (2023) CVE-2022-2353 vulnerability report.

Traditional LINE graph embedding algorithms cannot be used directly on unstructured data, and we propose the keywords-LINE algorithm for unstructured data processing. An unstructured representation of vulnerability report description is created by extracting keywords through natural language processing techniques. Structured parts can be extracted from the reports. Figure 4 shows the process of constructing a vulnerability report graph.

The number of vulnerabilities in the AVD is large and diverse, and the data processing includes the following steps:

1. Data screening and data cleaning. A small percentage of security vulnerability reports in the database contain incomplete information. In order to get as many complete reports as possible to ensure the quality of the dataset, we used a data screening process. Data screening ensures that the reports contain at least four or more vulnerability features. Data screening preserves relatively complete vulnerability reports and removes incomplete ones. Some complete vulnerability reports may have useless information, such as website links, which can be removed by regular expressions.

Figure 4. Vulnerability Report Graph Construction Process



2. Natural language processing of unstructured data. We used natural language processing techniques to extract keywords from vulnerability descriptions. We constructed a deactivation thesaurus to delete unimportant words in the vulnerability descriptions to improve the quality of text features. The vulnerability descriptions are partially subdivided into word sequences using a word segmentation technology, and the keywords are selected using the term frequency-inverse document frequency (TF-IDF) algorithm. TF-IDF is a widely used statistical technique in natural language processing. The core TF-IDF algorithm idea is to measure the significance of a word based on its frequency in one class and its frequency in others. When a word occurs frequently in a class of vulnerabilities that does not happen in other classes, it is selected as the keyword.

$TF_{i,j}$ denotes the word frequency, which is the word frequency of the word w_i in the same class of vulnerability description documents d_j. The frequency of each word in the same class of vulnerability documents is as follows in formula (1):

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

In formula (1), $n_{i,j}$ denotes the number of occurrences of word w_i in the same type of vulnerability description document d_j, and $\sum_k n_{k,j}$ denotes the sum of the occurrences of all words in the same type of vulnerability description document d_j.

IDF_i indicates the prevalence of a keyword. If the fewer documents containing the term w_i, the larger the IDF_i, the better the category differentiation ability of the term. IDF_i is as follows in formula (2):

$$IDF_i = \log \frac{\left|D\right|}{\left|\left\{j : w_i \in d_j\right\}\right| + 1} \tag{2}$$

In formula (2), $\left|D\right|$ denotes the total number of all vulnerability description documents, $\left|\left\{j : w_i \in d_j\right\}\right|$ denotes the total number of vulnerability documents containing the word w_i, and one is added to ensure that the denominator is not zero.

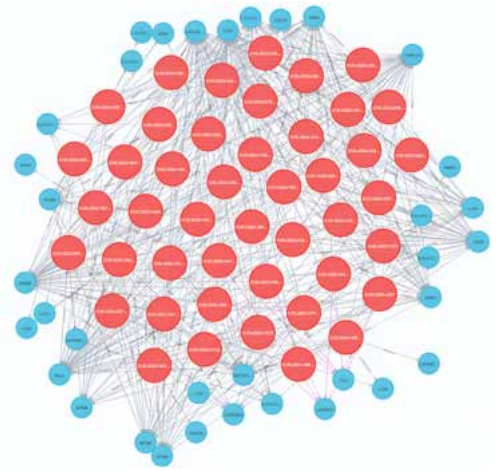
TF-IDF is calculated by formula (3):

$$TF - IDF = TF_{i,j} \times IDF_i \tag{3}$$

After calculating the TF-IDF value of word w_i, the TF-IDF value is used as the weight of word w_i. Then, the words which have high weights are selected as the keywords of the vulnerability description.

3. Entity identification, feature identification, relationship extraction, and knowledge fusion. The report graph consists of two types of entities. Entities comprise vulnerability and feature entities.

Figure 5. Part of the Vulnerability Report Graph Display



The vulnerability entity node has vulnerability a name and attributes (CVE-ID, vulnerability source, disclosure time); the feature entity describes the vulnerability attributes such as vulnerability category, CWE-ID as a categorization label, CVSS3 score, attack path, attack complexity, privilege requirement, type of affected software, and keywords of vulnerability description. It is necessary to analyze and fuse the relationship between vulnerability entities and feature entities to provide data support for constructing graphs.

4. Construction of vulnerability report graph. The vulnerability report graph is composed of vulnerability entities (V), relationships (R) and feature entities (F). Neo4j is a high-performance graph database specialized in storing, managing, and querying graph data. In this paper, we constructed vulnerability entity nodes and feature entity nodes on Neo4j using cypher language and stored attributes in vulnerability entity nodes. We constructed the relationship between the vulnerability entity nodes and the feature entity nodes, which is represented in the graph in the form of vulnerability entity (V)-[: relationship(R)]→feature entity (F). Among them, V represents a specific vulnerability, denoted by CVE-ID; R represents the name of the vulnerability feature, and F represents the specific attribute values corresponding to the feature. For example, if the attack path of the CVE-2022-2353 vulnerability is the network, we can represent it in the diagram as CVE-2022-2353(V)-[: Attack Path(R)]→Network(F).

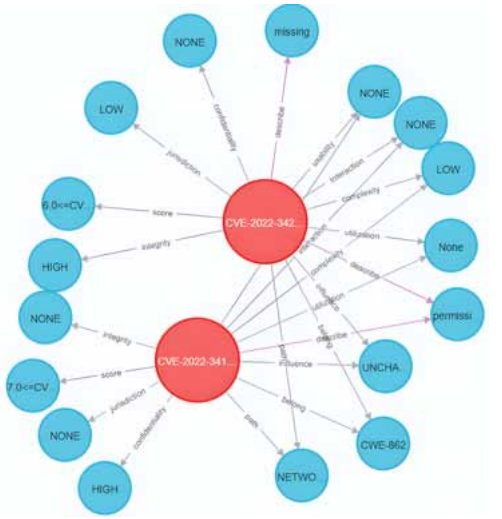
Organizing the vulnerability report diagram in this way makes it possible to visualize the complex relationships between vulnerability entities and features. Figure 5 shows the part of the vulnerability report graph. The large red nodes distributed throughout the center of the graph represent vulnerability entity nodes; the small blue nodes distributed throughout the edge of the graph represent feature entity nodes.

LINE Diagram Embedding Algorithm

LINE aims to represent each target node's local and global structure in the graph in a low-dimensional vector space. The core idea of the LINE graph embedding algorithm is to optimize the local structure vectors by using first-order similarity and optimize the global structure vectors by using second-order similarity to vectorize the high-dimensional graph nodes into a low-dimensional space. Following is the explanation of each concept.

First-order similarity: If there exists a directly connected edge between two nodes, the two nodes have high first-order similarity (or 0 if it does not exist), which should be close to each other in the

Figure 6. Graph of Two Vulnerability Nodes



embedding space. As shown in Figure 6, the two vulnerability nodes and the feature nodes in the report graph, the entity node CVE-2022-34201 has the keywords "missing" and "permission" by TF-IDF extracted from its vulnerability description. However, the entity node CVE-2022-34180 has only the keyword, "permission." Therefore, we can assume that the CVE-2022-34180 vulnerability has a substantial first-order similarity to the keyword "permission," but the first-order similarity of the keyword "missing" is 0.

Second-order similarity: If there are no directly connected edges between two nodes, they have many neighboring nodes in common. The two points have a high second-order similarity. Second-order similarity preserves the global structure well, which complements that first-order similarity ignores the global structure. Two nodes possessing higher second-order similarity should also behave close to each other in the second-order embedding space. For example, in Figure 6, the vulnerability entity node CVE-2022-34180 and the vulnerability entity node CVE-2022-34201 have no directly connected edges (first-order similarity is 0). These two entities have common neighbor nodes, such as they have the same usability, interaction, complexity, utilization, influence, path, and vulnerability description, and keyword "permission," so these two nodes have high second-order similarity.

First-Order Similarity Optimization Objective

The LINE algorithm first randomly initializes the embeddings that define each node. The joint probability between each two points is as formula (4):

$$p_1\left(v_x, v_y\right) = \frac{1}{1+\exp\left(-\vec{u}_x^T \cdot \vec{u}_y\right)} \tag{4}$$

where $\vec{u}_x$ is the low-dimensional vector representation of the vertex x.

The empirical distribution probability is as formula (5):

$$\hat{p}_1 = \frac{w_{xy}}{W} \quad \mathrm{W} = \sum_{(x,y)\epsilon E} w_{xy} \tag{5}$$

After obtaining the joint probability distribution and the empirical probability, KL measures the distance between two probability distributions in the same event space:

$$O_1 = d\left(\hat{p}_1\left(\cdot,\cdot\right), p_1\left(\cdot,\cdot\right)\right) \tag{6}$$

In formula (6), $d\left(\cdot,\cdot\right)$ is the distance between the two distributions; the KL distance is set as a loss function for training. The KL distance is calculated in formula (7) after ignoring the constant term:

$$O_1 = -\sum_{(x,y)\epsilon E} w_{xy} log p_1\left(v_x, v_y\right) \tag{7}$$

Second-Order Similarity Optimization Objective

For second-order similarity, each node is composed of two embedding vectors. One is the vector representation of the node itself, and the other is the vector representation of the node when it is the context vertex of other nodes. The probability of generating a neighbor node y given the node *x* is:

$$p_2\left(v_y | v_x\right) = \frac{\exp\left(\vec{u}_y'^T \cdot \vec{u}_x\right)}{\sum_{k=1}^{|V|} \exp\left(\vec{u}_k'^T \cdot \vec{u}_x\right)} \tag{8}$$

In formula (8), $\left|V\right|$ is the number of contextual vertices.

The optimization objective is calculated by formula (9):

$$O_2 = \sum_{x\in V} \lambda_x d(\hat{p}_2\left(\cdot | v_x\right), p_2(\cdot |\ v_x)) \tag{9}$$

where λ_x is the importance factor of the control node, let $\lambda_x = d_x$, d_x is the out degree of point *x*, and the empirical distribution probability is calculated as:

$$\hat{p}_2(v_y \mid v_x) = \frac{w_{xy}}{d_x} \tag{10}$$

In formula (10), w_{xy} is the edge power of *(x,y)*.

Neglecting the constant term is calculated by formula (11):

$$O_2 = -\sum_{(x,y)\in E} w_{xy} log p_2(v_y \mid v_x) \tag{11}$$

Using negative sampling to optimize the objective function improves the efficiency of computing second-order similarity. When it optimizes parameters using gradient descent, the edge weight

coefficient w_{xy} is multiplied by the gradient, which makes it challenging to select the appropriate learning rate if the variance of the edge weights in the graph. It is significant. By default, all edge weights are set to 1, solving the learning rate selection problem.

In this experiment, we assigned embedding vectors randomly to initialize each vulnerability entity node and feature entity node in the report graph. Subsequently, the LINE graph embedding algorithm trains the vector representation of each node. As the number of iterations increases, the LINE graph embedding model continuously updates the vector representations of the nodes by capturing the local and global similarities between the nodes. Vulnerable entity nodes with first-order and second-order similarities are presented in the embedding vector space.

Neighboring Nodes' Attention Mechanism

The neighboring nodes' attention mechanism considers the relationship between a node and its neighboring nodes to improve the representation of the node. The LINE graph embedding algorithm initializes the mapping of graph nodes into low-dimensional embedding vectors by calculating the attention weights between a node and its neighboring nodes. These weights indicate the level of attention a node pays to its neighboring nodes, and different nodes pay different levels of attention to their neighboring nodes.

We used dot product to calculate the similarity of the second-order embedding vectors of node *i* and node *j*. The keywords-LINE model introduces the neighbor node attention mechanism, which can capture the correlation and similarity between nodes more accurately. Thus, similar vulnerability embedding vectors can be represented more similarly in order to improve the effectiveness of vulnerability classification.

OVR Random Forest Multi-Classification Algorithm

OVR random forest multiclass classifier is an extended classification method for solving multi-classification problems. The algorithm creates *K*-independent binary classifiers for a dataset with *K* categories. In each classifier, only one category *i* is considered a positive example, and the other *K-1* categories are negative examples. Each binary classification constructs an independent random forest classifier. In the training phase, the training set is input to the random forest classifier to train. In the prediction phase, each random forest classifier receives prediction samples for prediction and determines the final multi-classification result based on voting. The algorithm decomposes the multi-classification problem into multiple classification problems, with each subproblem solved by a separate random forest model.

The vulnerability entity node vectors after first-order similarity and second-order similarity embedding are input into the OVR random forest multiclass classifier for classification. During the training phase, it creates the same number of independent random forest sub models with the total number of vulnerability categories. The purpose of each sub model is to distinguish the vulnerabilities of the corresponding category from the other categories as much as possible. In the testing phase, the test dataset embedding vectors are fed into each trained random forest multiclass classifier sub model for prediction, and each sub model provides a classification prediction for that vulnerability node and votes, then obtains the final classification result.

This classification strategy makes the model more independent in learning each category's features and helps classify categories with unbalanced data effectively. It is more suitable for the recognition, verification, and classification of multiple categories and multiple identities by training unique classifiers for each category. However, OVR random forest multiple classifiers tend to add extra computation overhead and increase the training time of the model.

Compared with other methods, our method adopts the vulnerability report graph to express the structured and unstructured information. The report graph can better handle the many-to-many and multilevel relationships between entities and model the complex structure of vulnerability

information more accurately. Keywords-LINE can embed the complicated local and global structures of the graph into vector space in a large-scale dataset. With the introduction of the neighbor node attention mechanism, keywords-LINE can capture the similar global structure of the graph better. OVR random forest classifier can train the unique vector characteristics of each type of vulnerability to better distinguish it from other vulnerabilities.

EXPERIMENT AND RESULTS

Vulnerability Dataset

Security vulnerability data has the characteristics of large data volume, diverse vulnerability features, distinguishing difficulty, and often requiring manual detection. Vulnerability reports can provide a detailed description of vulnerability characteristics, but there are unstructured and multisource heterogeneity situations.

AVD not only includes vulnerability reports from MITRE's CVE Vulnerability Database but also works closely with China National Computer Network Emergency Response Center (2023), China National Vulnerability Database (2023), and China National Vulnerability Database of Information Security (2023). So, it has the advantages of authoritative vulnerability reports, fast updates, and a large amount of vulnerability data. It is worth noting that a significant portion of the publicly available vulnerability features in the AVD are presented in a structured form, as exemplified by the information provided in Table 1. The structured expression improves the usability of the database and the efficiency of data analysis and processing. The extraction of keywords from the unstructured vulnerability description part compensates for the lack of specificity of structured vulnerability data and enriches its diversity. The AVD contains many different kinds of data, comprehensively collects the information of vulnerability reports, and conducts vulnerability classification efficiently and accurately.

The experiments described in this research use public security vulnerability reports from June 1, 2022, to December 31, 2022, in the AVD. Across the eleven types of vulnerabilities with the highest number of vulnerability reports, 4,058 vulnerability reports were selected, and 52,982 features were extracted.

Figure 7 shows the percentage of these eleven types of vulnerabilities among all classes of vulnerabilities in the last five years. With the detailed data for each type of vulnerability presented in Table 2, we observed a significant difference in the number of vulnerabilities. Conducting experiments with this data-imbalanced dataset genuinely reflects the real-world situation where the number of vulnerabilities is uneven. Table 3 shows the keywords for each type of vulnerability selected by the TF-IDF algorithm.

According to our methodology, ternaries (vulnerability entities, feature entities, and relationships) are extracted from the vulnerability report to construct the vulnerability report graph and stored in the Neo4j graph database. The vulnerability report graph consists of 4,058 vulnerability entity nodes, 109 classes of feature nodes, and 52,982 relationships.

Experimental Results and Conclusion Analysis

In this section we verify the validity of our method. To evaluate classification results, the accuracy, $F1_{micro}$, $F1_{macro}$, and $F1_{weighted}$ are introduced.

Classification Results Comparison of Experimental Results of Several Graph Embedding Algorithms

In the experiment, we used the DeepWalk graph embedding algorithm, node2vec graph embedding algorithm, and LINE graph embedding algorithm combined with different OVR classifiers to conduct comparative experiments to compare the performances of different graph embedding models in processing vulnerability report graphs. Table 4 shows the specific evaluation metrics.

Table 1. Vulnerability Information Display

Data Type	Vulnerability Features
Structured data	CVE-ID
	Vulnerability name
	Vulnerability source
	CWE name
	CWE-ID
	NVD published date
	CVSS3 score
	Utilization
	Patch status
	Attack path
	Attack complexity
	Privileges required
	Scope
	User interaction
	Availability impact
	Confidentiality impact
	Integrity impact
	Affected software types
	Affected software vendors, products
	Affected software version, affected area
Unstructured data	Vulnerability description
	solution suggestions

Figure 7. The Number of Eleven Types of Vulnerabilities Per Year as a Percentage

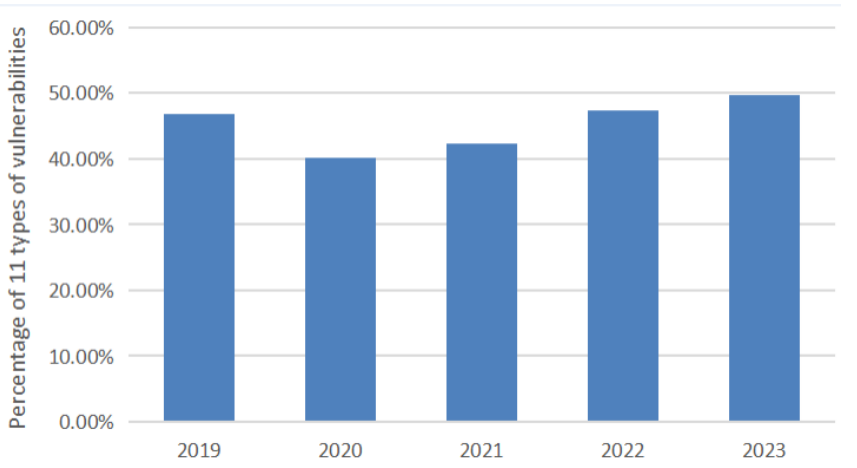


Table 2. Detailed Data of Each Type of Vulnerability

Vulnerability	CWE-ID	Number of Vulnerable Entities	Number of Feature Entities
Out-of-bounds write	CWE-787	774	9694
Improper neutralization of input during web Page Generation (cross-site scripting)	CWE-79	1114	15201
Improper neutralization of special elements used in an SQL command (SQL injection)	CWE-89	657	9093
Cross-site request forgery (CSRF)	CWE-352	253	3347
Missing authorization	CWE-862	139	1721
Improper neutralization of special elements used in a command (command injection)	CWE-77	136	1733
Improper input validation	CWE-20	190	2649
Improper limitation of a pathname to a restricted directory (path traversal)	CWE-22	201	2574
Out-of-bounds read	CWE-125	282	3086
Improper authentication	CWE-287	149	1827
Use after free	CWE-416	163	2057

By comparing the LINE model with the DeepWalk and node2vec models in the same classifier, it can be seen that classification accuracy of the LINE model is slightly higher than the other two models. The LINE method can discover similar characteristics between similar vulnerabilities by better preserving the local and global structure of the vulnerability report graph and embedding the graph nodes into the vector space using the optimized loss function. At the same time, the LINE model performs better in larger datasets, and allows similar vulnerabilities to be more closely represented in vector space.

Under the same graph embedding algorithm, the OVR random forest classifier is more accurate. When combined with graph embedding tasks, the OVR random forest classifier shows the best results compared with other OVR classifiers. It can deal with high-dimensional features and large-scale nonlinear data and train in the multiclass binary more effectively. It is able to capture the similar expression properties of similar vulnerabilities in the vector space and distinguish the differences in the vector space more accurately.

Classification Results Comparison of Various Keywords-LINE Structure

In order to compare various keywords-LINE structures, we offer the comparison of keywords-LINE only first-order similarity model, only second-order similarity model, and complete keywords-LINE model. Table 5 shows the experimental results.

From Table 5, first-order similarity is more critical than second-order similarity. This indicates that when the vulnerability report graph is too sparse and the number of neighboring nodes is small, the second-order similarity may become inaccurate, and even the performance of the model using only LINE second-order similarity is not as good as that of the DeepWalk model and the node2vec model. The keywords-LINE model combining first-order similarity and second-order similarity has

Table 3. Keywords for Each Type of Vulnerability

CWE-ID	Keywords	CWE-ID	Keywords
CWE-787	overflow	CWE-79	site
	stack		scripting
	buffer		XSS
	write		stored
	out of bounds		plugin
CWE-89	SQL	CWE-352	CSRF
	injection		forgery
	php		request
	parameter		admin
CWE-862	permission	CWE-77	command
	missing		injection
	authentication		cgi
CWE-20	validation	CWE-22	traversal
	input		file
	tensorflow		absolute path
CWE-125	read	CWE-416	free
	out-of-bounds		use
	memory		memory
CWE-287	server	authentication	
	access	Bypass	

Table 4. Comparison of Vulnerability Classification Results With Many Graph Embedding Models

Algorithm	$F1_{micro}$	$F1_{macro}$	$F1_{weighted}$	Accuracy
DeepWalk+logistic regression	0.909	0.863	0.907	0.909
DeepWalk+decision tree	0.921	0.873	0.920	0.921
DeepWalk+KNN	0.944	0.925	0.943	0.944
DeepWalk+random forest	0.948	0.932	0.946	0.948
Node2vec+logistic regression	0.920	0.880	0.918	0.920
Node2vec+decision tree	0.925	0.900	0.925	0.925
Node2vec+KNN	0.947	0.921	0.947	0.947
Node2vec+random forest	0.949	0.930	0.948	0.949
LINE+logistic regression	0.918	0.891	0.917	0.918
LINE+decision tree	0.926	0.878	0.925	0.926
LINE+KNN	0.946	0.923	0.946	0.946
LINE+random forest	0.958	0.940	0.958	0.958

Table 5. Comparison of the Impact of Model Completeness on Vulnerability Classification Results

Algorithm	$F1_{micro}$	$F1_{macro}$	$F1_{weigted}$	Accuracy
Keywords-LINE(1st)	0.949	0.931	0.949	0.949
Keywords-LINE(2nd)	0.838	0.824	0.838	0.838
Keywords-LINE(1st+2nd)	0.958	0.940	0.958	0.958

Table 6. Comparison of the Impact of Classification Results With Neighbor Nodes' Attention Mechanism

Algorithm	$F1_{micro}$	$F1_{macro}$	$F1_{weigted}$	Accuracy
Keywords-LINE(1st)	0.949	0.931	0.949	0.949
Keywords-LINE(2nd) without adding attention mechanism	0.838	0.824	0.838	0.838
Keywords-LINE(2nd) adding attention mechanism	0.913	0.905	0.912	0.913
Keywords-LINE(1st+2nd) without adding attention mechanism	0.958	0.940	0.958	0.958
Keywords-LINE(1st+2nd) adding attention mechanism	0.975	0.968	0.975	0.975

the best performance. This result demonstrates fully that the first-order similarity and second-order similarity of the keywords-LINE graph embedding algorithm are complementary, namely, local structure and the global structure compensate for each other.

The Experiments of Nodes' Attention Mechanism

In order to improve the classification accuracy, we incorporated a neighbor node's attention mechanism, which assigns attention weights by calculating the similarity of the second-order embedding vectors of node *i* and node *j*. After introducing the attention mechanism of neighbor nodes, we further evaluated the performance of the keywords-LINE model. Table 6 shows the impact on the keywords-LINE model results after adding the neighbor nodes' attention mechanism.

To understand the specific changes for each category after the introduction of the neighbor nodes' attention mechanism in more detail, we introduced confusion matrices illustrated in Figures 8 and 9.

In Figure 8, we find that some vulnerability categories, such as CWE-77 and CWE-20, perform poorly in classification effectiveness without the neighbor nodes' attention mechanism. It may be due to the relatively small number of vulnerabilities in these categories, which are not distinguished effectively from other vulnerabilities, thus leading to a degradation in classification performance. With the introduction of the second-order attention mechanism, however, keywords-LINE can capture the similarity of this category better in the global structure and represent them more accurately in the vector space. As shown in Figure 9, we observed that the accuracy improved significantly.

Comparison of Classification Results Between Traditional LINE and Keywords-LINE Models

Table 7 compares the effect of the presence or absence of keywords on the evaluation metrics of the model through experiments.

Table 7 shows the comparison of the classification results with and without keywords. The keywords-LINE algorithm is better than the traditional LINE graph embedding algorithm and the

Figure 8. No Neighbor Nodes' Attention Mechanism

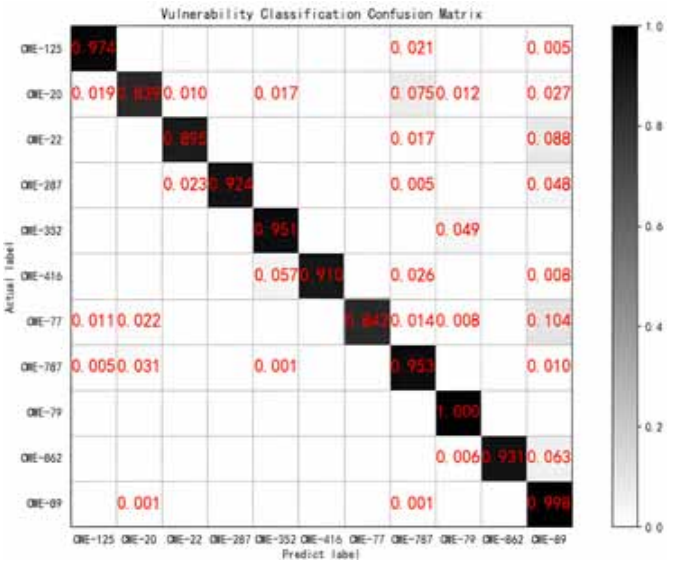


Figure 9. Joined Neighbor Nodes' Attention Mechanism

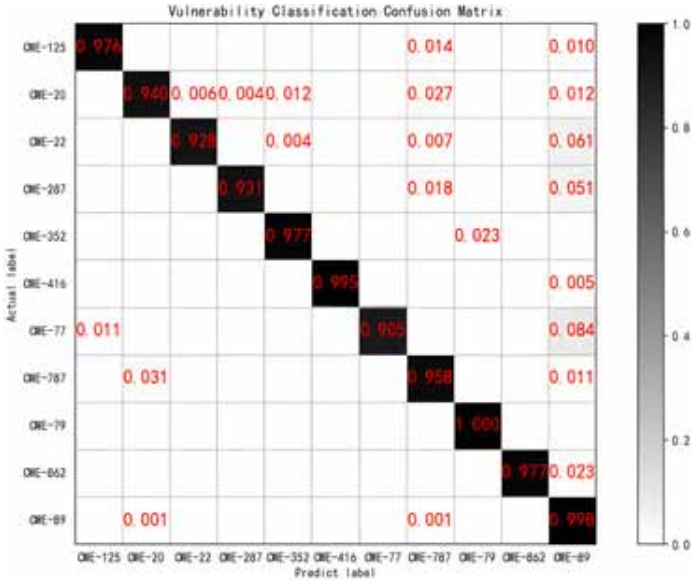


Table 7. Comparison of Vulnerability Classification Results With and Without Keywords

Algorithm	$F1_{micro}$	$F1_{macro}$	$F1_{weighted}$	Accuracy
Only TF-IDF vulnerability describing keywords	0.961	0.925	0.960	0.961
LINE (without keywords)	0.956	0.942	0.956	0.956
Keywords-LINE	0.975	0.968	0.975	0.975

Table 8. Comparison With Other Vulnerability Classification Experimental Results

Method	$F1_{micro}$	$F1_{macro}$	$F1_{weighted}$	Accuracy
LDA	0.938	0.882	0.934	0.938
LSA	0.905	0.889	0.903	0.905
TF-IDF	0.961	0.925	0.960	0.961
VCGERG	0.975	0.968	0.975	0.975

only TF-IDF keywords (we only used the part of vulnerability description in the report), specifically the *macro-F1*.

Compared with the traditional LINE graph embedding algorithm, the keywords-LINE algorithm has unique features for each class of vulnerabilities, improving the evaluation metrics.

Unstructured Vulnerability Description Features Representation Experiments

The experiment compares with the LDA (Aljedaani et al., 2020), LSA (Alperin et al., 2020), TF-IDF, and our method. We used the LDA, LSA, and TF-IDF methods to extract the keywords in the description part of the vulnerability report and used the OVR random forest classifier to classify them. The evaluation metrics are shown in Table 8.

VCGERG is our method, which has the highest classification accuracy of 0.975. Different methods of extracting keywords may differ due to the other methods of calculating keyword weights. The LDA keyword extraction method tries to find the subject words behind similar vulnerability description documents and may perform better in discovering the potential topics of the text. The LSA downscaling technique based on singular value decomposition captures the likely semantic structure in the text and, to a certain extent, considers the semantic relationships between words. However, the TF-IDF method of extracting keywords more comprehensively considers the word frequency of the class of vulnerabilities and the inverse document frequency of the entire corpus of vulnerability descriptions. Hence, the keywords extracted by the TF-IDF method better reflect the unique characteristics of each class of vulnerabilities.

By comparing the keywords extracted by LDA and TF-IDF, we found that LDA tends to extract more repetitive keywords. For example, for three categories of vulnerabilities, CWE-20, CWE-22, and CWE-125, LDA extracts the same keyword "file," which reduces the overall classification accuracy.

Compared with extracting keywords from vulnerability descriptions for classification using only the keyword extraction method, our VCGERG model shows significantly higher improvement on $F1_{macro}$ than on $F1_{micro}$. This phenomenon indicates that the VCGERG model can ensure high classification stability even when there is a significant gap in the data for each type of vulnerability.

The Effect of Using Negative Sampling Loss

In this study, we used negative sampling loss optimization. Figure 10 shows the effect of using different negative sampling loss, maximizing the inner product loss and Jaccard loss.

The selection of the optimization loss function affects the quality of the embedding vectors learned by the keywords-LINE model directly. During the training process, the keywords-LINE model provides feedback optimization by minimizing the negative sampling loss function. This advantage of using a negative sampling loss function to optimize keywords-LINE may be related to the following reasons: Negative sampling loss uses an independent negative sampling strategy to select negative samples, which reflects the differences between positive and negative samples,

Figure 10. Comparison of the Impact of Different Loss Functions on Vulnerability Classification Results

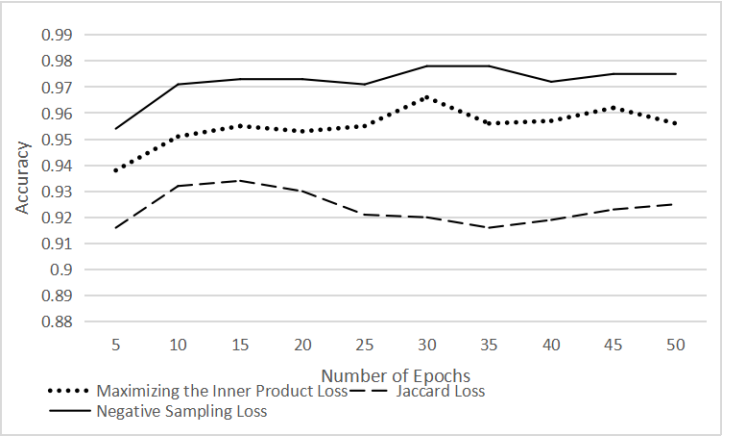
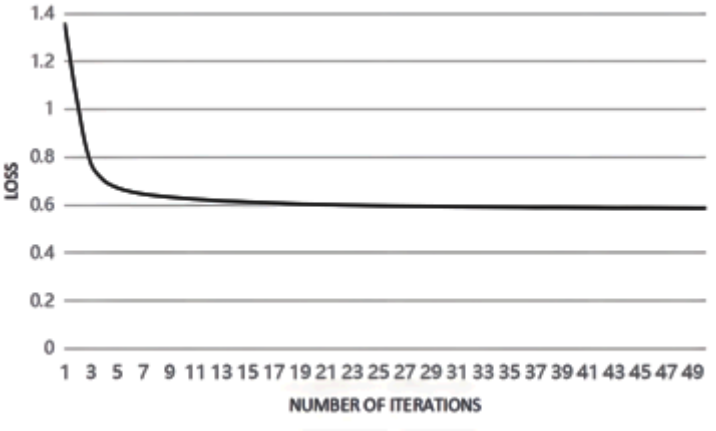


Figure 11. Loss Curve Graph



distinguishes the class of samples from other classes of samples, and improves the accuracy of classification. Compared with LINE models using other loss functions, LINE models using negative sampling loss have higher classification stability. It is because the negative sampling loss function uses a random negative sample sampling strategy during the training process, comparing each positive sample with a negative sample, improving the robustness. Compared with other loss functions, it reduces the dependence on specific negative samples. It is less susceptible to interference from noise and outliers.

The loss curve is plotted to determine the strength of the model. In Figure 11, the loss value decreases to convergence as the number of training times increases. It indicates that the optimization algorithm has brought the loss function to a locally optimal solution. The dataset performs better on the model, avoiding overfitting while maintaining a specific generalization ability to provide more accurate results.

CONCLUSION AND FUTURE WORK

In this paper, we propose the VCGERG method to represent vulnerability reports. The VCGERG model constructs a graph of reports and introduces the neighboring node attention mechanism to represent the structural embedding vectors of the vulnerability graph based on the keywords-LINE graph embedding algorithm. The method can capture not only local information but also the global structure of the graph more accurately. We conducted experiments on the AVD vulnerability database using the OVR random forest multi-classifier. The model accuracy has the highest classification accuracy of 0.975, improved by 1.7%. At the same time, the VCGERG model improves the $F1_{macro}$ and $F1_{micro}$ metrics more significantly, by 8.6% and 7.9%, respectively. This means that our method can deal with the challenge of the imbalance of various types of large-scale vulnerability information effectively.

Overall, the VCGERG method can be used as a powerful auxiliary tool for vulnerability management systems to correct possible human errors and improve the accuracy of vulnerability detection and classification. The VCGERG methodology can also be used for vulnerability scanning and corporate software security assessment, helping security organizations and enterprises manage and classify vulnerabilities more effectively and enhance the overall security of society. By improving the efficiency and accuracy of vulnerability detection, management, and classification, the VCGERG method can reduce the risks of economic loss and data leakage brought about by malicious vulnerability issues in governments and companies and help build a more secure and reliable digital society. In addition, since the VCGERG method performs more accurately and consistently in handling large-scale multi-classification imbalance data. It also has the potential to be extended to other classification prediction areas.

In summary, vulnerability reports are valuable, and more attention needs to be paid to them. In future work, we plan to link the vulnerability codes with corresponding vulnerability reports for more precise analysis and classification of vulnerabilities.

CONFLICTING INTERESTS

All the authors of this article declare there are no competing interests.

FUNDING STATEMENT

The work was sponsored by the Beijing Natural Science Foundation of China Grant No.4232021. The National Nature Science Foundation of China Grant No.62232016.

PROCESS DATES

Received: November 3, 2023, Revision: February 9, 2024, Accepted: February 9, 2024

CORRESPONDING AUTHOR

Correspondence should be addressed to Xiaoyi Zhao; 2108550021043@stu.bucea.edu.cn

REFERENCES

Alibaba Cloud Vulnerability Database (AVD). (2023, March 21). *Vulnerability DB*. https://avd.aliyun.com/

Aljarf, A., Zamzami, H., & Gutub, A. (2023). Is blind image steganalysis practical using feature-based classification? *Multimedia Tools and Applications*, ●●●, 1–34.

Aljedaani, W., Javed, Y., & Alenezi, M. (2020). LDA categorization of security bug reports in chromium projects. In *Proceedings of the 2020 European Symposium on Software Engineering*. doi:10.1145/3393822.3432335

Alperin, K. B., Wollaber, A. B., & Gomez, S. R. (2020). Improving interpretability for cyber vulnerability assessment using focus and context visualizations. In *Proceedings of the 2020 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE. doi:10.1109/VizSec51108.2020.00011

Altalhi, S., & Gutub, A. (2021). A survey on predictions of cyber-attacks utilizing real-time twitter tracing recognition. *Journal of Ambient Intelligence and Humanized Computing*, *12*(11), 1–13. doi:10.1007/s12652-020-02789-z

Aota, M., Kanehara, H., Kubo, M., Murata, N., Sun, B., & Takahashi, T. (2020). Automation of vulnerability classification from its description using machine learning. In *Proceedings of the 2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. doi:10.1109/ISCC50000.2020.9219568

China National Computer Network Emergency Response Center. (2023, March 21). *China national computer network emergency response center.* https://www.cert.org.cn/

China National Vulnerability Database. (2023, March 21). *China national information security vulnerability database.* https://www.cnvd.org.cn/

China National Vulnerability Database of Information Security. (2023, March 21). *National information security vulnerability library.* http://www.cnnvd.org.cn/

Common Weakness Enumeration (CWE). (2023, March 21). *CWE top 10 KEV weaknesses*. https://cwe.mitre.org/

Costa, J. C., Roxo, T., Sequeiros, J. B., Proenca, H., & Inacio, P. R. (2022). Predicting CVSS metric via description interpretation. *IEEE Access : Practical Innovations, Open Solutions*, *10*, 59125–59134. doi:10.1109/ACCESS.2022.3179692

CVE. (2017). *Common vulnerabilities & exposures*. https://cve.mitre.org/cve/

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. doi:10.1145/2939672.2939754

Gutub, A., Shambour, M. K., & Abu-Hashem, M. A. (2023). Coronavirus impact on human feelings during 2021 Hajj season via deep learning critical Twitter analysis. *Journal of Engineering Research*, *11*(1), 100001. doi:10.1016/j.jer.2023.100001

Han, Z., Li, X., Liu, H., Xing, Z., & Feng, Z. (2018). Deepweak: Reasoning common software weaknesses via knowledge graph embedding. In *Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. doi:10.1109/SANER.2018.8330232

Han, Z., Li, X., Xing, Z., Liu, H., & Feng, Z. (2017). Learning to predict severity of software vulnerability using only vulnerability description. In *Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. doi:10.1109/ICSME.2017.52

Hemalatha, J., Sekar, M., Kumar, C., Gutub, A., & Sahu, A. K. (2023). Towards improving the performance of blind image steganalyzer using third-order SPAM features and ensemble classifier. *Journal of Information Security and Applications*, *76*, 103541. doi:10.1016/j.jisa.2023.103541

Hu, Y. T., Wang, S. Y., Wu, Y. M., Zou, D. Q., Li, W. K., & Jin, H. (2023). A Slice-level vulnerability detection and interpretation method based on graph neural network. *Journal of Software, 34*(6), 0.

Huang, J., Han, S., You, W., Shi, W., Liang, B., Wu, J., & Wu, Y. (2021). Hunting vulnerable smart contracts via graph embedding based bytecode matching. *IEEE Transactions on Information Forensics and Security*, *16*, 2144–2156. doi:10.1109/TIFS.2021.3050051

National Institute of Standards and Technology (NIST). (2023a). *National vulnerability database*. https://nvd.nist.gov/

NIST. (2023b). *National vulnerability database: CVE-2022-2353 detail*. https://nvd.nist.gov/vuln/detail/CVE-2022-2353

NIST. (2023c). *National vulnerability database: CVE-2022-32893 detail.* https://nvd.nist.gov/vuln/detail/CVE-2022-32893

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. doi:10.1145/2623330.2623732

Roy, P. K., Saumya, S., Singh, J. P., Banerjee, S., & Gutub, A. (2023). Analysis of community question-answering issues via machine learning and deep learning: State-of-the-art review. *CAAI Transactions on Intelligence Technology*, *8*(1), 95–117. doi:10.1049/cit2.12081

Sufi, F. K., Alsulami, M., & Gutub, A. (2023). Automating global threat-maps generation via advancements of news sensors and AI. *Arabian Journal for Science and Engineering*, *48*(2), 2455–2472. doi:10.1007/s13369-022-07250-1

Sun, H., Cui, L., Li, L., Ding, Z., Hao, Z., Cui, J., & Liu, P. (2021). VDSimilar: Vulnerability detection based on code similarity of vulnerabilities and patches. *Computers & Security*, *110*, 102417. doi:10.1016/j.cose.2021.102417

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. doi:10.1145/2736277.2741093

Wartschinski, L., Noller, Y., Vogel, T., Kehrer, T., & Grunske, L. (2022). VUDENC: Vulnerability detection with deep learning on a natural codebase for Python. *Information and Software Technology*, *144*, 106809. doi:10.1016/j.infsof.2021.106809

Zou, D., Wang, S., Xu, S., Li, Z., & Jin, H. (2019). μVulDeePecker: A deep learning-based system for multiclass vulnerability detection. *IEEE Transactions on Dependable and Secure Computing*, *18*(5), 2224–2236. doi:10.1109/TDSC.2019.2942930

Yashu Liu received his Ph.D. degree in Computer Science from Beijing Jiaotong University, P.R. China. Now, she works in the department of computer science and technology, Beijing University of Civil Engineering and Architecture. She is an associate professor and supervisor. She majors in data mining, network security. She has published many scientific papers and patents.

Xiaoyi Zhao is currently pursuing his master's degree with the School of Electrical and Information Engineering, Beijing University of Civil Engineering and Architecture, Beijing, China. His research interests focus on cyber security.

Xiaohua Qiu is currently pursuing his master's degree with the School of Electrical and Information Engineering, Beijing University of Civil Engineering and Architecture, Beijing, China. Her research interests focus on cyber security.

Han-Bing Yan, Ph.D., professor-level senior engineer, doctoral supervisor, main research areas are network security, machine learning, and computer graphics.