Journal of Electronic Research and Application, 2025, Volume 9, Issue 5
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

A Binary Vulnerability Similarity Detection Model Based on Deep Graph Matching

Yangzhi Zhang*

School of Artificial Intelligence, Zhejiang Dongfang Polytechnic, Wenzhou 325000, Zhejiang, China

**Author to whom correspondence should be addressed.*

Abstract: To enhance network security, this study employs a deep graph matching model for vulnerability similarity detection. The model utilizes a Word Embedding layer to vectorize data words, an Image Embedding layer to vectorize data graphs, and an LSTM layer to extract the associations between word and graph vectors. A Dropout layer is applied to randomly deactivate neurons in the LSTM layer, while a Softmax layer maps the LSTM analysis results. Finally, a fully connected layer outputs the detection results with a dimension of 1. Experimental results demonstrate that the AUC of the deep graph matching vulnerability similarity detection model is 0.9721, indicating good stability. The similarity scores for vulnerabilities such as memory leaks, buffer overflows, and targeted attacks are close to 1, showing significant similarity. In contrast, the similarity scores for vulnerabilities like out-of-bounds memory access and logical design flaws are less than 0.4, indicating good similarity detection performance. The model's evaluation metrics are all above 97%, with high detection accuracy, which is beneficial for improving network security.

Keywords: Network security; Word vectors; Graph vector matrix; Deep graph matching; Vulnerability similarity

Online publication: October 21, 2025

1. Introduction

The richness of modern life is highly dependent on the progress of the Internet, which allows people to meet their needs for shopping, learning, and socializing. However, the vast amount of data generated by the Internet of Things also increases the likelihood of network attacks. The main cause of network attacks is software vulnerabilities, which can lead to the theft of a large amount of information from computers, system slowdowns, and even complete system failures [1]. In general, software vulnerabilities are analyzed using both source code and binary code. However, the majority of software is released in binary form. When two or more binary files have vulnerabilities that are similar in functionality and code logic, this is referred to as binary vulnerability similarity. The presence of similar binary vulnerabilities can rapidly spread network attacks and increase the difficulty of defense. Therefore, detecting the similarity of binary vulnerabilities is of great significance.

The main idea of deep graph matching technology is to detect the similarity of two or more binary vulnerabilities by matching the network feature graphs and network feature words. This study is an innovation in vulnerability detection technology, integrating deep graph matching with vulnerability similarity detection. It plays a key role in the detection process and lays a solid foundation for future developments in this field.

2. Vulnerability similarity detection model construction

2.1. Word vectorization

The prerequisite for inputting network data samples into the detection model is to complete the vectorization conversion of network information feature data. This vectorization conversion is achieved through word embedding, which maps words into a real-valued vector space, transforming words expressed in natural language into vectors or matrices that computers can understand. Similarly, transforming images into vectors or matrices is known as image embedding. The Continuous Bag-of-Words (CBOW) model is an important component of the embedding model, capable of generating word vectors by predicting the input itself based on the semantic features of preceding words and the features of following words. In this study, the CBOW model is selected to vectorize the features of network data samples. The first step is to segment the network data samples into multiple words based on instructions and features, and then train the CBOW model to obtain the word and image vector matrices [2]. The vector conversion process typically uses a corpus with dimensions ranging from 130 to 220. Since the vocabulary of the network data samples to be converted is smaller than that of ordinary text classification, this study converts the first eight non-numeric features of the network data samples into 20-dimensional vectors, with the remaining numeric features placed after the 20-dimensional feature vectors. Viewing the sample as a code segment, each code segment contains 50 instructions, which can be converted into a 176-dimensional vector. Therefore, the vector dimension of a code segment is 50×176. Finally, the vectorized network data features are input into the trained vulnerability similarity detection model to complete the vulnerability similarity detection.

2.2. Vulnerability similarity detection model

Long Short-Term Memory (LSTM) is an important deep learning method. The advantage of LSTM is its ability to extract contextual information from sequential data. The binary vulnerability similarity detection based on deep graph matching involves feature extraction of the embedded words and images to analyze their intrinsic associations and similarities. The structure of the vulnerability similarity detection model based on LSTM is shown in **Figure 1**.

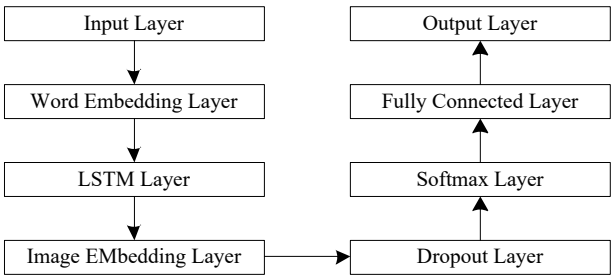

Figure 1. Structure of the vulnerability similarity detection model based on LSTM

After the network information data is input, the Word Embedding layer is responsible for vectorizing the data. The LSTM layer is in charge of mining and extracting the contextual associations of the quantized network

information word vector matrix, and it also performs feature extraction on the word vectors from the Word Embedding layer and the embedded images to analyze their intrinsic associations and similarities. The Dropout layer then randomly deactivates neurons in the results analyzed by the LSTM layer to reduce the probability of overfitting. To transform the raw output of the neural network into a probability distribution problem, the Softmax layer maps the output results analyzed by the LSTM layer to the interval (0,1). Finally, the fully connected layer completes the detection result output with a dimension of 1.

3. Vulnerability similarity detection based on deep graph matching

3.1. Word embedding

Word embedding can express network information data samples as real-valued vectors, and these real-valued vectors can significantly increase the similarity between word vectors of contextually similar words. This similarity can greatly reduce the distance between two word vectors in the vector space [3]. In the Word Embedding layer, assume that a word in a data sample is *w*, and there are *c* words before and after it. The word vectors of these *c* words are input into the CBOW model for training and the output of the word vector for *w*. After the CBOW model is trained, the output word vectors have clear relevance and analogy. Moreover, the word vectors can clearly express the meaning of the word itself as well as the intrinsic relationships and similarities between words.

3.2. Graph embedding

Assume that a 3-layer control flow graph *g* is input into the detection model, and all vertices *v* within the graph are related to a feature vector k_i. Under training conditions, the features of all vertices will be updated to form a new feature vector $\vec{g}$. Through iterative aggregation, the embedded graph is obtained $\vec{g}$. When the number of iterations is *t*, the new feature $\mu_i^{(t)}$ is obtained by combining the vertex features and the graph structure features. The *t*+1-th iterative update formula for the updated feature vector μ_i of all vertices is:

$$\mu_i^{(t+1)} = F\left(k_{\upsilon_i}, \sum_{j \in N_{(\upsilon_i)}} \mu_j^{(t)}\right), \forall \upsilon_i \in V \tag{1}$$

Here, the *i*-th vertex, the set of all vertices, and the set of neighbors of the vertex are denoted by v_i, V, and $\backslash N_{(\upsilon_i)}$, respectively. F represents the nonlinear mapping, and the mapping formula is:

$$F\left(k_{\upsilon_i}, \sum_{j \in N_{(\upsilon_i)}} \mu_j^{(t)}\right) = tanh\left(W_1 k_{\upsilon_i} + \sigma\left(\sum_{j \in N_{(\upsilon_i)}} \mu_j^{(t)}\right)\right) \tag{2}$$

Here, F represents the matrix of dimension k_i (vertex feature vector) × dimension of the graph embedding vector. The hyperbolic tangent function and the nonlinear function are denoted by *tanh* and σ, respectively. The formula for the nonlinear function of the *n*-layer fully connected neural network is:

$$\sigma(y) = P_1 \times ReLU\left(P_2 \times \cdots ReLU\left(P_n \times y\right)\right) \tag{3}$$

Here, P_1, P_2, and P_n are the hyperparameter matrices of dimensions *n* (graph embedding vector dimension) × graph embedding vector dimension for the 1st, 2nd, and *n*-th layers of the neural network graph embedding vectors, respectively. $ReLU(\cdot)$ represents the output value of the rectified linear unit function, and it satisfies $ReLU(\cdot) = max\{0, m\}$. After *T* iterations, the embedded graph $\vec{g}$ is:

$$\vec{g} = W_2 \sum_{v_i \in V} \mu_i^{(T)} \tag{4}$$

Here, W_2 represents the matrix of dimension × graph embedding vector dimension × embedding vector dimension, used to transform the final graph embedding vector.

3.3. LSTM

Long Short-Term Memory (LSTM) networks are an extension of the standard Recurrent Neural Network (RNN) and are highly effective in detecting sequential feature relationships. RNN can dynamically simulate the input, output, and hidden states of network nodes. Suppose the input, output, and hidden state are represented by x_t, y_t, and h_t, respectively. Since the current network information is based on the previous network information, the formula for the hidden state of a standard RNN is:

$$h_t = f_h\left(x_t, h_{t-1}\right) \tag{5}$$

Here, f_h represents the state transition function of the network node, and h_{t-1} represents the hidden state of the network node at the previous moment. The formula for the network node output state is:

$$y_t = f_0\left(h_t\right) \tag{6}$$

Here, f_0 represents the output function of the network node. The input state x_t of the network node can be regarded as the sequence element of the RNN network. By combining the current input state x_t with the output h_{t-1} of the previous hidden layer state, the hidden layer state output h_t can be obtained. However, the RNN network can only preserve short-term input sequence information of the network nodes. Therefore, it is necessary to build upon the RNN network with LSTM to preserve long-term input sequence information of the network nodes. LSTM also has the capability to extract contextual information from sequential data, which enables effective feature extraction of the embedded words and images, and analysis of their intrinsic associations and similarities [4]. The LSTM network is a standard cell structure, where all cells update the current cell state c_t and hidden state h_t based on the output of the previous moment, and then output the updated states. Therefore, the LSTM network can effectively preserve the current cell state and the previous cell states. The forget gate f_t, input gate i_t, and output gate o_t contribute to the memory-preserving function of the LSTM network.

The detection of binary vulnerability similarity first requires converting binary code into a sequence and inputting the result into the LSTM network. The forget gate of the LSTM network can preserve long-term memory of the sequence cell state. In other words, the forget gate selects information relevant to the similarity detection of the current word feature vector and the corresponding graph feature vector from the previous cell state for selective memory. The current input state x_t can be regarded as the code feature being analyzed, while the output h_{t-1} of the previous hidden layer state can be seen as a previously saved code sequence fragment. Since not all word and graph code features are useful for the entire detection, the forget gate needs to set the corresponding memory unit value to 0 to forget the word and graph code features. However, if the detection model is detecting a vulnerability pattern with a loop, the forget gate needs to set the corresponding memory unit value close to 1 to promote the memory of the word and graph code features, thereby achieving vulnerability detection. The forget gate is responsible for forgetting code features unrelated to vulnerability detection, while the input gate judges the new word and graph code features and their value, storing them in long-term memory. During this period, the vulnerability similarity detection model can form a feature representation that meets the requirements

of vulnerability patterns and good refinement based on the sequence of word and graph code features [5,6]. The candidate state vector containing high-value information is formed by compressing the network node input state x_t through the tanh function. Then, the nonlinear function outputs the code features x_t of the word and graph being analyzed and the previously saved code sequence fragment x_t, forming a valve vector that can determine the importance of all information in the candidate memory. In the LSTM network, the output gate is responsible for selecting the sequence information that can represent the time step from long-term memory as the feature vector of the vulnerability word and graph for output. The valve vector formed by the nonlinear function output of the input gate analyzes the outputtable vulnerability feature vectors in long-term memory, and then the tanh function activates the output of the vulnerability word and graph feature vectors. The final output feature vector formula is:

$$h_t = o_t \times tanh(c_t) \tag{7}$$

Assume that a binary function graph embedding is transformed into a basic block feature sequence (BB_1,BB_2,…,BB_T), which is input into the LSTM network and processed through three gates to form a new hidden state. The formula for calculating the binary function graph embedding vector is:

$$V_{func} = h_T \tag{8}$$

Here, h_T is the hidden state vector generated by the LSTM network during feature sequence processing, which also represents the semantic features of the binary function graph embedding.

Assume that the word embedding vector and the graph embedding vector obtained through the binary function graph embedding vector calculation method are V_{word} and V_{image}, respectively. The detection result is determined by the vulnerability similarity score, which is calculated using the following formula:

$$Similarity(word, image) = cos(\theta) = \frac{V_{word} \cdot V_{image}}{\|V_{word}\| \|V_{image}\|} = \frac{\sum_{i=1}^{n} V_{word}^{i} V_{image}^{i}}{\sqrt{\sum_{i=1}^{n} (V_{word}^{i})^2} \sqrt{\sum_{i=1}^{n} (V_{image}^{i})^2}} \tag{9}$$

Here, $cos(\theta)$ represents the dot product of the word embedding vector V_{word} and the graph embedding vector V_{image} . $\|V_{word}\|$ and $\|V_{image}\|$ are the magnitudes of the word embedding vector V_{word} and the graph embedding vector V_{image}, respectively. The closer the final vulnerability similarity score is to 0, the more dissimilar the two or more vulnerabilities are. Conversely, the closer the vulnerability similarity score is to 1, the higher the similarity between multiple vulnerabilities.

4. Experimental analysis

4.1. Vulnerability similarity detection

Before training the detection model, the corresponding training parameters need to be set. The learning rate and the number of epochs are set to 0.0001 and 50, respectively. The training set consists of 3-layer control flow graphs compiled from two sources: the same source function and different source functions. One graph is selected from each of the two sources of 3-layer feature control flow graphs, denoted as g_1 and g_2. Here, g and g_1 are images generated by compiling the same source function, while g and g_2 are images generated by compiling different source functions. The training samples ⟨ g, g_1⟩ and ⟨ g, g_2⟩ are labeled as +1 and -1, respectively.

Due to the uncertainty of images across different epochs, the training data varies from epoch to epoch. To enhance the comparability of the experiments, all the data trained in different epochs are re-randomly input into

the detection model for training. The graph embedding dimension is set to 128, the embedding depth to 2, and the number of iterations to 5. The SVM detection model, KNN detection model, and GGNN detection model are selected as the benchmarks for comparison with the LSTM-based vulnerability similarity detection model used in this paper. The ROC curves of different methods are compared, as shown in **Figure 2**. The ROC curves of the test set and the training set for the LSTM vulnerability similarity detection model almost overlap, and it is evident that the LSTM model outperforms the other detection models. The AUC of the LSTM vulnerability similarity detection model is 0.9721. In contrast, after training with the same data, the AUC of the SVM detection model drops from 0.878 in the test set to 0.725, the KNN detection model drops from 0.776 in the test set to 0.602, and the GGNN detection model drops from 0.648 to 0.524. It is clear that the vulnerability similarity detection model proposed in this paper has the most stable performance.

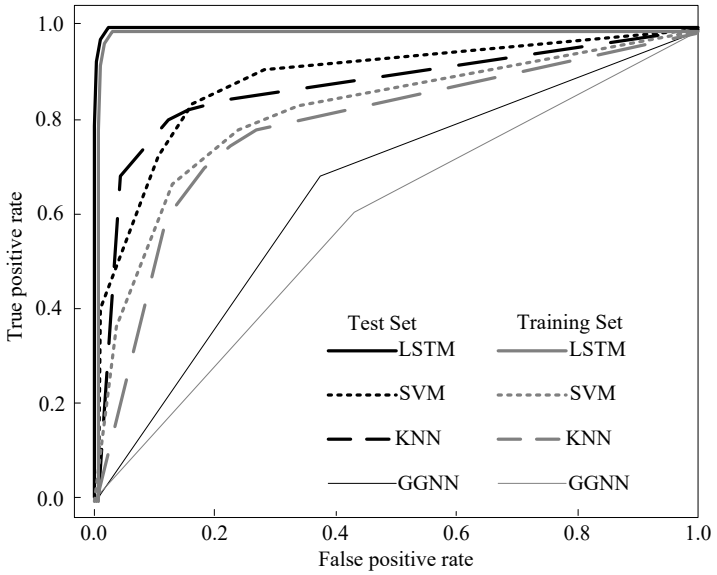

Figure 2. Comparison of ROC curves for different methods

4.2. Detection effect

The binary vulnerability similarity detection model based on deep graph matching proposed in this paper is used to detect the similarity of vulnerabilities that have been determined to be similar. By comparing the detection results with the actual situation, the detection effect of the vulnerability similarity detection model in this paper is determined. The detection effect of the model is shown in **Table 1**.

The closer the similarity score is to 0, the more dissimilar the two or more vulnerabilities are. The closer the similarity score is to 1, the higher the similarity between multiple vulnerabilities. For the three types of vulnerabilities—memory leaks, buffer overflows, and targeted attacks—the similarity scores obtained using the detection model in this paper are close to 1, indicating a clear similarity between the vulnerabilities. For the two types of vulnerabilities—out-of-bounds memory access and logical design flaws—the similarity scores are less than 0.4, indicating no similarity between the vulnerabilities. These results are consistent with the actual situations of the five types of vulnerabilities. Therefore, the binary vulnerability similarity detection model based on deep graph matching can effectively detect the similarity of vulnerabilities and enhance network security.

Table 1. Detection effect of the model

Program vulnerability categories	Actual situation	Similarity detection results	
		Vulnerability similarity scores	Similarity determination
Memory leak	Vulnerability is the same	0.98	Similarity vulnerability
Buffer overflow	Vulnerability is the same	0.99	Similarity vulnerability
Out-of-bounds memory access	Different functionality	0.28	Non-similarity vulnerability
Logical design flaw	Function after patching	0.37	Non-similarity vulnerability
Targeted attack	Vulnerability is the same	0.95	Similarity vulnerability

4.3. Evaluation metrics

To verify that the binary vulnerability similarity detection model based on deep graph matching proposed in this paper can effectively improve the detection accuracy, this section will analyze the SVM detection model, the KNN detection model, the GGNN detection model, and the detection model proposed in this paper. Four commonly used evaluation metrics—accuracy, recall, precision, and F1-score—will be used to assess the proposed detection model.

Assume that the data features in the dataset are divided into two classes: positive and negative. *TP* (True Positive) represents the number of positive instances correctly predicted as positive by the algorithm. *TN* (True Negative) represents the number of negative instances correctly predicted as negative. *FP* (False Positive) represents the number of negative instances incorrectly predicted as positive. *FN* (False Negative) represents the number of positive instances incorrectly predicted as negative.

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{10}$$

$$recall = \frac{TP}{TP + FN} \tag{11}$$

$$precision = \frac{TP}{TP + FP} \tag{12}$$

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{13}$$

The evaluation results for the SVM, KNN, and GGNN detection models are all below 85% in terms of accuracy, recall, precision, and F1-score, indicating that their overall detection accuracy is not satisfactory (**Table 2**). In contrast, the binary vulnerability similarity detection model based on deep graph matching proposed in this paper achieves evaluation metrics above 97%, demonstrating its ability to accurately detect the similarity of binary vulnerabilities.

Table 2. Evaluation results for different detection models

Detection model name	Accuracy (%)	Recall (%)	Precision (%)	F1 (%)
SVM detection model	81.39	79.47	82.95	78.48
KNN detection model	83.68	79.76	81.54	80.45
GGNN detection model	82.54	84.35	83.41	83.49
Binary vulnerability similarity detection model based on deep graph matching	97.33	98.52	97.46	97.86

5. Conclusion

This paper introduces deep graph matching technology into program vulnerability detection. By using deep learning to mine the word vector features and graph vector features of vulnerabilities, similarity detection is performed on vulnerabilities within different binary programs. The LSTM network is primarily used to extract the feature vectors of word embeddings and graph embeddings, analyzing their intrinsic associations and similarities. The results show that the AUC of the LSTM-based vulnerability similarity detection model in this paper is 0.9721. In contrast, the AUC of the SVM detection model drops to 0.725 after training, the KNN detection model drops to 0.602, and the GGNN detection model drops to 0.524. The evaluation metrics of the SVM, KNN, and GGNN detection models are all below 85%, while the evaluation metrics of the vulnerability similarity detection model in this paper are all above 97%. The binary vulnerability similarity detection model based on deep graph matching not only has stable performance but also can accurately detect the similarity of binary vulnerabilities, effectively safeguarding network security.

Funding

Special Project Funded by Tsinghua University Press: "Engineering Drawing and CAD" Course Construction and Textbook Development

Disclosure statement

The author declares no conflict of interest.

References

[1] Yang S, Xu Z, Xiao Y, et al., 2023, Towards Practical Binary Code Similarity Detection: Vulnerability Verification via Patch Semantic Analysis. ACM Transactions on Software Engineering and Methodology, 32(6): 1–29.

[2] Li L, Ding S H H, Tian Y, et al., 2023, VulANalyzeR: Explainable Binary Vulnerability Detection with Multi-Task Learning and Attentional Graph Convolution. ACM Transactions on Privacy and Security, 26(3): 1–25.

[3] Zhu Y, Lin G, Song L, et al., 2023, The Application of Neural Network for Software Vulnerability Detection: A Review. Neural Computing and Applications, 35(2): 1279–1301.

[4] Wen X C, Gao C, Ye J, et al., 2023, Meta-Path Based Attentional Graph Learning Model for Vulnerability Detection. IEEE Transactions on Software Engineering, 50(3): 360–375.

[5] Tang M, Tang W, Gui Q, et al., 2024, A Vulnerability Detection Algorithm Based on Residual Graph Attention Networks for Source Code Imbalance (RGAN). Expert Systems with Applications, 238: 122216.

[6] Yan X, Sun M, Han Y, et al., 2023, Camouflaged Object Segmentation Based on Matching–Recognition–Refinement Network. IEEE Transactions on Neural Networks and Learning Systems, 35(11): 15993–16007.