Research Article

A Software Vulnerability Detection Method Based on Complex Network Community

Chun Shan,1 Yinghui Gong,1 Ling Xiong,1 Shuyan Liao,2 and Yuyang Wang1

1Beijing Key Laboratory of Software Security Engineering Technology, School of Computer Science & Technology, Beijing Institute of Technology, Beijing 100081, China
2School of Cyberspace Science & Technology, Beijing Institute of Technology, Beijing 100081, China

Correspondence should be addressed to Chun Shan; sherryshan@bit.edu.cn

Received 11 December 2021; Revised 11 March 2022; Accepted 14 March 2022; Published 11 May 2022

Abstract

To find out whether there is any vulnerability in software programs where conditional judgment is ignored, this article proposes a software vulnerability detection method based on complex network community. First, the method abstracts the software system into a directed weighted graph by using the software algebraic component model and then preprocesses the directed weighted graph to get a complex network graph. Then, by using the partition algorithm, the complex network graph is divided into the communities, and the key nodes in communities are found by nRank algorithm. Finally, the graph of the key nodes with high influence is matched with the complex network graph that has been preprocessed. In order to evaluate the effectiveness of the community partition algorithm and the nRank algorithm, comparative experiments are carried out on two datasets. The experimental results show that the community partition algorithm is better than the comparison algorithm in precision, recall, and comprehensive evaluation index, and the nRank algorithm is closer to the result of degree centrality measurement index than the PageRank algorithm and the LeaderRank algorithm. The spring-shiro-training project is used to verify the vulnerability detection method based on complex network community, and the results show that the method is effective.

1. Introduction

As the function of software increases, its complexity will increase [1]. These vulnerabilities can affect the functionality and performance of the system, posing a threat to software security. Hackers can exploit software vulnerabilities to carry out network attacks and cause enormous losses [2]. Software security issues require further attention. Vulnerability detection can be divided into static analysis, dynamic analysis, and hybrid analysis according to the analysis method. Static analysis [3–5] refers to analyzing source code syntactically or semantically without executing the program. It is an effective method, but it has a high false positive rate. Dynamic analysis [6] is to obtain and analyze the dynamic information generated during the operation of an executable code. Compared with dynamic analysis, static analysis has a good advantage in the preprocessing of source code. Hybrid detection [7, 8] combines static analysis and dynamic analysis, which can provide more powerful functions and improve accuracy. In addition, many studies use traditional machine learning techniques [9–11]. However, traditional machine learning-based solutions require experts to define features clearly. The experience and knowledge of experts limits the quality of manually determined features. Deep learning technology [12, 13] can automatically extract features, freeing experts from tedious feature engineering tasks.

With the deepening of network theory research, scientists gradually adopted the view of complex network to study complex systems in nature. A complex system is a network composed of many of closely connected and interacting units. The theory of complex network is used to study protein-protein interaction in biological research [14]. The idea of using complex network to analyze the topological structure of a software system was first started by researchers in statistical physics and complex system science [15]. Myers et al. [16] found that most of the topological structures of

Copyright © 2022 Chun Shan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
software systems have the relevant characteristics of complex networks such as "small world" and "scale-free." Literature [17] regards software system as a complex network graph composed of algebraic components and connection relations between them.

For the study of complex networks, researchers have gone deep into the study of community structure. To study the community structure, the commonly used community mining algorithms are mainly divided into two kinds: one is the global community mining algorithm, and the other is the local community mining algorithm. The global community mining algorithm is mainly based on complex network information for community analysis. The spectral clustering algorithm [18] represents the graph with a specific matrix (Laplace matrix), decomposes it into the feature vector, then uses the traditional clustering method, and then clusters the feature vectors using traditional clustering methods into communities. The GN algorithm [19] adopts the idea of segmentation. According to the degree that the edge does not belong to the community, delete the edges between the communities until all the edges that do not belong to the community are deleted. The fast Newman algorithm [20] is a further improvement in the GN algorithm. It treats each node as a community and then continuously merges communities along the direction that allows the modularity to move in one direction. ASC algorithm [21] proposes a new spectral clustering method to detect communities in attribute networks. The weights are calculated according to the cosine similarity of node attributes and assigned to the edges of the graph. This method improves the detection accuracy and can apply to large-scale network. The algorithm complexity of the global community mining algorithm is high, and it is difficult to get all kinds of information about complex networks. In this case, the researchers turned to local community mining algorithms. The local community mining algorithm was first proposed by Aaron Clauset, who proposed the R algorithm [22] based on local modularity. This algorithm performs local mining by maximizing the increment in local modularity, where R refers to the ratio of the number of all edges inside the community to the number of all edges outside the community. Kamal Berahmand et al. [23] proposed an algorithm called ECES (expanding core nodes using extended similarity) that simultaneously detects all communities in a graph using the local information of the graph and used a new measure of local centrality to detect core nodes. Whether it is the global community mining algorithm or the local community mining algorithm, the size of the complex network, the connection between the nodes, and other factors may cause the final community to be different. Based on the local community mining algorithm, this article evaluates the centrality of nodes in the community through three common evaluation indexes such as degree centrality, betweenness centrality, and closeness centrality and proposes the average centrality of nodes based on the three evaluation indexes to reduce the deviation. By finding the central node of each community in the complex network, and then according to the characteristics of the community structure, the neighbor nodes that are closely connected with the central node are classified into the community where they belong, and finally several communities are formed.

Research into node sorting in complex network also considers the importance of nodes in the whole complex network graph or in part of the complex network graph. It is mainly considered from the following four aspects: First, the sorting method of neighbor nodes based on nodes. For example, the importance of a node in a graph is measured by degree centrality. Its disadvantage is that it only considers the local information of the node. Second, the sorting method based on the paths between the nodes. For example, the importance of a node in the graph is measured by betweenness centrality. Third, the sorting method based on the feature vectors of the nodes, for example, PageRank algorithm [24]. The disadvantage of this algorithm is that the existence of dangling nodes will affect the accuracy of the results. Furthermore, the LeaderRank algorithm [25] is based on the PageRank algorithm, and a special node is inserted to form a strongly connected graph. The LeaderRank algorithm uses this special node to replace the jump probability in the PageRank algorithm, which is much faster than the PageRank algorithm. Fourth, the sorting method based on node removal and shrinkage. In this method, the nodes in the complex network graph are constantly deleted, which leads to the imbalance of the node relationship in the complex network graph. Its disadvantage is that it has a high computational complexity. The above four types of methods are basically designed based on the importance of any node in the complex network graph to the whole complex network or to a part of the complex network, but the mutual influence between the nodes in the graph is not considered. Based on the LeaderRank algorithm, this article introduces the similarity of nodes to judge the importance of nodes in part of the complex network graph and proposes a new sorting algorithm called nRank algorithm.

This article is divided into four parts. The first part introduces the current software vulnerability detection technology and complex network theory and explains the major research objectives of this study. The second part introduces the materials and methods used in this project. The software system is viewed from the perspective of complex network. The complex network graph is divided and sorted, and then graph matching is used to find the algebraic components that may have vulnerabilities in the software system. The third part is the experimental part, where an example of a software system and datasets are used to verify the method. It is proved that the method is effective. The fourth part is the summary of this article.

2. Materials and Methods

2.1. Selection of Vulnerabilities. In practical applications, unchecked parameters may cause software vulnerabilities. Such vulnerabilities may not cause operating errors under normal circumstances, but if they are attacked, they are likely to be executed in a non-original direction. The following Algorithm 1 is an example of what is the vulnerability of conditional judgment being ignored:
In this code, when the system calls the func() method, it must first determine whether the value of the parameter a is valid. If the parameter a is not checked when the func() method is called, the path will be lost. In this case, the attacker can use the parameter to generate some illegal parameters, which will bring serious consequences to the execution of the code.

2.2. Software Algebraic Component Model. Based on the software algebraic topology diagram [26], the software system can be divided into functional modules. The software system can be identified as composed of several functional modules, and the functional modules in the software system are abstracted into algebraic components. There may be relations of call, inclusion, and nesting among the functional modules in the software system, so there are also three relations among the algebraic components. The software system can be regarded as a set composed of abstract algebraic components and their connection relations.

2.2.1. Definition of Algebraic Component. To represent the functional modules in the software system with algebraic components, the definition of algebraic components is given. According to [27], the algebraic component is regarded as a six-tuple \( PM = (X, Y, D, C, c, \eta) \). A six-tuple is composed of two parts, including connecting parts and functional parts. X represents a set of logical behaviours, which is composed of multiple logical behaviours and is expressed as \( X = (x_1, x_2, \ldots, x_n) \). Here, the connection relations between the algebraic components can be divided into three types: inclusion, nesting, and call; C represents the controller, which is used to receive input data and transmit output data; D represents the internal data; \( \varsigma \) represents the aggregation operation. \( \eta \) represents the detector, and the detector is divided into two parts: the input discriminator and the output discriminator. The input discriminator is mainly aimed at the effective connection relationship, converts it according to the data and transmission format requirements of the algebraic component, and transmits the conversion result to the controller and the data storage. The output discriminator mainly controls the output to different algebraic components according to the input results of the controller, data memory, and arithmetic unit.

Any algebraic component can be represented by a six-tuple, and finally the software system is simplified into a complex network graph. The edges of a complex network graph are the connections between algebraic components. According to the relations between the modules in the software system, the operation relations among algebraic components can be divided into three kinds: call, inclusion, and nesting. The priority order is inclusion > nesting > call.

2.2.2. Generation of Software Topology Diagram. According to [27], the algebraic components are all stored in the xml file format, in which the associated tags and property information are defined. If there are any of the three connection relations among various algebraic components, it will be added to the property information. Then the algebraic components form a software algebraic topology diagram through the connection relations among the algebraic components. Finally, the software system is abstracted into a software algebraic topology diagram.

2.3. Preprocessing of Complex Network Graph. Because there may be over one connection relation among algebraic components, it also means that there is over one edge among nodes in the complex network graph. Such a graph needs to be preprocessed before it becomes the input data required by the community partitioning algorithm. Here are some basic concepts of graph theory that will be used.

(1) Frequent subgraph: In the graph dataset \( G = \{G_1, G_2, \ldots, G_n\} \), if the number of subgraphs of \( G \) containing \( H \) is greater than the product of the minimum support (user-specified threshold) and the number of subgraphs in graph database, then \( H \) is called frequent subgraph of \( G \).

(2) Maximal frequent subgraph: In all frequent subgraphs of a graph dataset, if the graph \( H \) is not a subgraph of any other frequent subgraph, then a graph \( H \) can be called a maximal frequent subgraph.

(3) Subgraph isomorphism: If a graph \( M \) is isomorphic to a subgraph of graph \( G \), it can be called a subgraph of graph \( G \), which can be obtained by adding or deleting edges in the subgraph of graph \( G \).

The preprocessing operation is as follows: EPDG (enhanced program dependency graph) is introduced to represent a function module. Assuming that there are two edge connections between the two nodes, select one edge, delete this edge, and add a virtual node. Then connect the virtual node to the original two nodes with two extra edges. If there are multiple edge connections between the two nodes, the multiple edges will be deleted, and repeat the above operation until there are no multiple edges.

2.4. Community Partition Algorithm. "Community structure" is an important structural feature of complex network graph. It means that the points in the community are closely connected with each other, while these points are loosely connected with the points outside the community [28]. The community partition algorithm introduced in this article is based on the idea of local mining algorithm, which is mainly divided into two steps. The first step is to get the set of central nodes of the community, and the second step is to divide the community through the idea of local optimization.

2.4.1. Find the Central Node Set. The central node refers to the relatively most important node in the community. This article adopts three centrality evaluation indexes: degree
centrality, betweenness centrality, and closeness centrality to measure the centrality of nodes.

(1) Degree Centrality. Degree centrality is a basic parameter that describes the structure of scale-free networks [29]. The ratio of the number of reachable paths between a node and other nodes to the total logarithm of that node and other nodes is called degree centrality, which is represented by DC. The calculation expression is shown in the following equation:

$$DC = \frac{d(v)}{n - 1},$$

where $n$ represents the number of nodes in the complex networks and $d(v)$ represents the degree of node $v$. The greater the value of DC, the greater the direct influence of the node on the complex networks, and the higher the probability that the node is the central node of the community.

(2) Betweenness Centrality. Betweenness centrality is the number of the shortest paths between any pair of nodes passing through a certain node in the path, which is represented by BC. The calculation expression is shown in the following equation:

$$BC = \sum_{x \neq y \neq v \in V} \frac{g_{xy}(v)}{g_{xy}}.$$  

where $V$ represents a set of nodes in a complex network, $g_{xy}(v)$ represents the number of the shortest paths between node $x$ and node $y$ in the path through node $v$, and $g_{xy}$ represents the number of all shortest paths between the nodes $x$ and $y$. Betweenness centrality shows the importance of a node in a local network.

(3) Closeness Centrality. Closeness centrality is the distance between one node and other nodes in a complex network, which is represented by CC. The calculation expression is shown in the following equation:

$$CC = \frac{n - 1}{\sum_{u=1}^{n} d(v, u)},$$

where $n$ represents the number of all nodes in the graph and $d(v, u)$ represents the number of edges in the shortest path from node $v$ to node $u$. Closeness centrality can measure whether a node is in a key position in a complex network. If the path $d(v, u)$ between a node and other nodes is shorter, then the CC of this node will be higher, showing that this node is likely to be the central node of the network.

(4) Average Centrality. No matter which single index is adopted to measure the centrality of nodes, there may be a large deviation. Therefore, this article calculates the average centrality of the center node through these three centrality evaluation indexes. The steps to find the average centrality of a node are as follows: Each centrality index of each node in the complex network graph is taken as the weight of this node, and each node is regarded as a scheme, then the average centrality $C_i$ of the $i$th node is calculated. The calculation expression is shown in the following equations:

$$r_{ij} = \frac{P_i(v_j)}{\max(P_i(v_j))}, \quad i = 1, 2, \ldots; j = 1, 2, \ldots,$$

$$C_i = \sum w_j r_{ij}, \quad i = 1, 2, \ldots; j = 1, 2, \ldots,$$

where $P$ represents the node as a set of schemes and $w_j$ represents the weight of the $j$ index of the node.

(5) Node Similarity. After calculating the degree centrality, betweenness centrality, and closeness centrality of nodes, the average centrality of nodes is calculated. Then the nodes are sorted according to their average centrality. The higher the average centrality is, the more important the node is in the community. Therefore, the top 15% of the nodes were selected after sorting to generate a candidate node set, then two different nodes in the candidate node set were selected each time and the similarity between the two nodes was calculated. If the value of similarity between the two nodes is high, the node with the lower value is deleted, and this operation is repeated until there are no more deletable nodes in the candidate node set. And the final node set is taken as the central node set of the community. Node similarity is used to describe the closeness of the relationship between the two nodes. It is the ratio of the number of common neighbor nodes of two nodes to the total neighbor nodes of two nodes. The calculation expression of the similarity $S(v_i, v_j)$ of node $v_i$ and $v_j$ is shown in the following equation:

$$S(v_i, v_j) = \begin{cases} \frac{|N(v_i) \cap N(v_j)|}{|N(v_i)| - 1}, & N(v_i) \neq 1, \\ 1, & N(v_j) = 1, \end{cases}$$

where $N(v_i)$ represents the set of all adjacent nodes of the node $v_i$. $v_j$ is the neighbor node of $v_i$. If $N(v_j)$ is equal to 1, it means that the node $v_j$ has only one neighbor node $v_i$. At this time, define the similarity $S(v_i, v_j)$ equal to 1. If $N(v_j)$ is not equal to 1, it means that node $v_j$ is only a neighbor node in the set of neighbor nodes of node $v_j$, then $|N(v_i) \cap N(v_j)|$ represents the number of common neighbor nodes of two nodes, and $|N(v_i)| - 1$ represents the number of neighbors of the node $v_i$ minus an edge between the nodes $v_i$ and node $v_j$.

(6) Steps of Finding the Central Node Set.

Step 1: Calculate the average centrality of the node based on the degree centrality, the betweenness centrality, and the closeness centrality of the node.

Step 2: Sort the nodes by the descending order of the average centrality of nodes and select the top 15% nodes to generate a candidate node set.

Step 3: Select the two non-repeated nodes in the candidate node set each time and calculate the similarity between the two nodes.

Step 4: If the similarity between the two nodes is great, delete the node with the lower value. Repeat this operation until there are no more removable nodes in the...
candidate node set and make such a node set the central node set of the community.

Step 5: If the similarity is not greater than the threshold, then select the central node set to join the network.

Here, the algorithm flowchart is shown in Figure 1.

In the process of implementing the algorithm to find the central node set, it is assumed that the degrees of the nodes in the candidate node set have been calculated. To represent the relationship between the nodes, the adjacency table is chosen here to store the serial number id of each node and sort the nodes in ascending order. For node \( v_i \), the degree of any of its neighbors can be obtained by the BFS algorithm, so that the comparison of the degrees can be used to determine whether node \( v_i \) itself is a local central node, and at the same time, the compared neighbor nodes need to be marked to avoid using them again, resulting in performance redundancy. Assuming that there are \( n \) nodes in the candidate node set: each node needs to perform the BFS algorithm, so the time complexity is \( O(n) \); each node needs to traverse one or more of its neighbor nodes, so the time complexity is also \( O(m + e) \); and the time complexity of calculating the similarity between the two nodes each time is \( O(1) \), so the total time complexity is \( O(n \times m) \).

### 2.4.2. Divide the Community

According to the characteristic that the community structure is a complex network with common characteristics, each central node with certain common characteristics will be divided into a set. According to the idea of local optimization, the neighboring nodes that meet the requirements are divided into a set each time until the partition is completed. Each set formed in this way is a community.

1. **The local module goodness of the community.** \( LC_0 \) can judge the quality of each community after a complex network is divided into several communities. The calculation expression is shown in the following equation:

\[
LC_0 = \frac{E_{in0}}{E_{out0}},
\]

where \( E_{in0} \) represents the total number of edges connected between any two nodes in the community and \( E_{out0} \) represents the total number of edges connected between any node outside the community and any node within the community. Now, one neighbor node of a node was added based on the original community. The \( LC \) of the community obtained is shown in the following equation:

\[
LC = \frac{E_{in0} + E_{in}}{E_{out0} + E_{out}},
\]

where \( E_{in} \) represents the increase in the number of edges connected between any two nodes in the community after joining a neighbor node and \( E_{out} \) represents the increase in the number of edges connected between any node outside the community and any node within the community after joining a neighbor node. After joining any neighbor node of a certain central node in the community, the expression of \( \Delta LC \) of the community is shown in the following equation:

\[
\Delta LC = LC - LC_0,
\]

where \( \Delta LC \) is used to judge the importance of any neighbor node of a certain central node of a certain community in the complex network to the community. \( \Delta LC \) is used to determine whether to add this neighbor node to the community where the central node is located.

2. **Steps of the Community Partition Algorithm.** The process of community partition algorithm adopts the idea of local optimization for each central node in the community. Here,

![Flowchart](Image)

**Figure 1:** Flowchart of finding the central node set algorithm.
a central node is taken as an example to illustrate the steps of the algorithm.

Step 1: Given a set without elements, take any central node from the obtained central node set and put it into this set, making the set a candidate node set containing a central node.

Step 2: Calculate the neighbor node set of the central node. Specifically, calculate the local module goodness increment of each neighbor node around the central node.

Step 3: If the local module goodness increment of a neighbor node is greater than the set threshold, then the neighbor node will be added to the candidate node set where the center node is located.

Step 4: If the local module goodness increment of the neighbor node is lower than the set threshold, then the neighbor node will be skipped. Then continue to select the next neighbor node from the neighbor node set of the central node until there is no neighbor node larger than the set threshold.

For the central node set, the process will be repeated until the last central node in the central node set is processed. After executing the local mining on all the central nodes, the obtained communities cannot be directly used as the final communities, they cannot be used as the input data for the next operation. There are two special cases that need to be dealt with. One is the case where there are nodes in the complex network that are not in any community. This type of node can be directly summarized to the node candidate set corresponding to the central node when the local module goodness increment of the node is highest. The other is the case where one node appears in two or more communities simultaneously in a complex network. For this type of node, the current $\Delta LC$ is compared with the last reserved $\Delta LC$ each time. If the current $\Delta LC$ is greater than the last reserved $\Delta LC$, the current $\Delta LC$ will be assigned to the last reserved $\Delta LC$. The historical maximum value of $\Delta LC$ reserved last time is added to the candidate node set where the current center node is located. Otherwise, the $\Delta LC$ retained last time remains unchanged, and the execution continues until all the nodes are in the candidate node set where the corresponding central node is located. The several sets formed at this time are the communities and are used as the input data for the next step.

Here, the algorithm flowchart is shown in Figure 2.

In the process of implementing the algorithm, it can be known from the previous step that the set of central nodes is known, assume that a central node is now taken out and put into an empty set, which constitutes the candidate set of nodes where the central node is located. If it is greater than the threshold, it will be added to the node candidate set where the central node is located; if it is less than the threshold, the neighbor node will be skipped and the next neighbor node will be taken, and the process will be repeated until the set of neighbor nodes can no longer be fetched. Assume that the number of nodes currently existing in the central node set is $n$ and each node has $m$ neighbor nodes.

Each node requires the BFS algorithm, so the time complexity is $O(n)$. Each node requires the BFS algorithm to traverse one or more of its neighbor nodes, so the time complexity is $O(m + e)$, and the time complexity of computing the $\Delta LC$ of each neighbor node is $O(1)$, so the total time complexity is $O(n \cdot m)$.

2.5. nRank Algorithm. In order to find vulnerabilities quickly, the communities are ranked in descending order according to their impact on the complex network.
Because the central node of the community is the most influential node in the community, the ranking of the community is converted to the ranking of the central node of the community. Based on the LeaderRank algorithm, this article introduces the similarity of nodes to judge the importance of nodes to some complex network graphs and presents a new ranking algorithm called nRank algorithm. In the nRank algorithm, the interaction between the nodes in the complex network graph is considered. The greater the similarity value of a node, the greater the importance of this node in the graph. The realization of nRank algorithm is mainly divided into four steps, the specific steps are:

**Step 1.** Through the LeaderRank algorithm, the LR of any node in the whole complex network is obtained. The calculation expression of LR is shown in the following equation:

\[
\begin{align*}
\text{LR}_i(0) &= \frac{1}{n} \\
\text{LR}_i(k) &= \sum_{j=0}^{n} a_{ij} \text{LR}_j(k-1), \quad k = 1, 2, 3, \ldots, \\
\text{LR}_i &= \text{LR}_k + \frac{\text{LR}_k(k)}{n}, \quad n = 1, 2, 3, \ldots,
\end{align*}
\]

where \( n \) represents the number of all nodes in the complex network and \( \text{LR}_i(k) \) represents the importance of the background node in the complex network graph at the \( k \)th time after multiple calculations.

**Step 2.** The similarity of nodes is introduced here to describe the core idea of nRank algorithm. The calculation expression is shown in the following equation:

\[
S = \gamma k_{ij}^\text{out} + (1 - \gamma)k_{ij}^\text{in},
\]

where \( 0 < \gamma < 1 \). Node \( v_j \) is a neighbor of node \( v_i \), \( k_{ij}^\text{out} \) represents the number of common connection nodes pointed to by nodes \( v_i \) and \( v_j \).

**Step 3.** In the second step, the similarity of the influence of any common connecting node on the two nodes can be calculated. Here, the similarity of all pairs is processed to get the R of the node. The calculation expression is shown in the following equation:

\[
R_i = \text{LR}_i \sum S_{k_{ij}^\text{out}} + 1 \text{LR}_j,
\]

where \( R_i \) represents the R value of node \( v_i \).

**Step 4.** Repeat the above steps until any node in the complex network graph has calculated the similarity between the nodes and the R of the nodes. Then, according to the R of the node to complete the ranking of the node. Then, sort the nodes according to their R.

Here, the algorithm flowchart is shown in Figure 3.

2.6. **Principle of Graph Matching.** After being processed by the node sorting algorithm, the frequent subgraphs mined are regarded as candidate rules, and then a set of rule graphs will be obtained, which are then matched with the original topological graph to find out those graphs with similar but different composition. These graphs can then be considered as graphs with possible vulnerabilities. Here, the graph matching algorithm used is based on a heuristic search algorithm [30].

3. **Results and Discussion**

3.1. **Experimental Environment.** This experiment is based on JDK 1.8.0 and Tomcat 7, using the Intellij IDEA 2019 development tool. The Gephi is used to process the Karate dataset and the Dolphins dataset.

3.2. **Experimental Data.** To verify the feasibility of the community partition algorithm and the nRank algorithm,
this article selects Karate dataset and Dolphins dataset. Moreover, the spring-shiro-training project is selected to verify the software vulnerability detection method based on a complex network community.

3.2.1. Karate Dataset. The Karate dataset contains 34 nodes and 78 edges and is shown in Figure 4.

3.2.2. Dolphins Dataset. The Dolphins dataset contains 62 nodes and 159 edges and is shown in Figure 5.

3.2.3. Spring-Shiro-Training Project. Spring-shiro-training is a permission system developed based on springmvc, spring, mybatis-plus, shiro, easyui, and Log4j2. The system function diagram of the spring-shiro-training project is shown in Figure 6.

3.3. Evaluation Index. For the verification of the community partition algorithm and nRank algorithm, this experiment selects three evaluation indicators: precision ($P$), recall ($R$), and comprehensive evaluation index ($F$).

3.3.1. Precision. In the experiment of software vulnerability detection method based on complex network community, the precision is used as the result evaluation index. Here, the precision is the ratio of the number of confirmed vulnerabilities among potential vulnerabilities discovered to the number of all potential vulnerabilities discovered.

3.3.2. Recall. The ratio of the number of nodes divided into pairs to the actual number of nodes in the community is called the recall.

3.3.3. Comprehensive Evaluation Index. Based on the precision and recall, a comprehensive evaluation of the community partition is carried out. The calculation expression is shown in the following equation:

$$F = \frac{2PR}{P + R} \quad (13)$$

3.4. Analysis of Experimental Results

3.4.1. Experimental Results of Community Partition Algorithm. To better explain the performance of the algorithm, the R algorithm given by Clauset is introduced here to perform a comparative experiment. The R algorithm is also based on the idea of local optimization. The community partition algorithm proposed in this article is carried out in two steps. The first step is to find the central node set, which requires the similarity and threshold between the candidate nodes to make judgments. The second step is to cut the community, which requires the $\Delta LC$ and threshold of the node to make judgments. Therefore, the same threshold is selected for both steps here. And it is measured by the three evaluation indicators $P$, $R$, and $F$.

When the threshold is 0.3, the result of the community partition algorithm based on the Karate dataset and Dolphins dataset is shown in Table 1. When the threshold is 0.32, the result of the community partition algorithm based on the Karate dataset and Dolphins dataset is shown in Table 2.

Based on the three evaluation indexes $P$, $R$, and $F$, the community partition algorithm is compared with the R algorithm. According to the experimental results shown in Tables 1 and 2, when the threshold is 0.3 or 0.32, the $P$, $R$, and $F$ of the community partition algorithm are better than the R algorithm for Karate dataset, and the $P$ of the community partition algorithm is lower than the R algorithm for the Dolphins dataset. However, both $R$ and $F$ are better than the R algorithm.

The Karate dataset can be divided into two communities using the community partition algorithm. The structure diagram after dividing into two communities is shown in Figure 7.

The Dolphins dataset can be divided into two communities using the community partition algorithm. The structure diagram after dividing into two communities is shown in Figure 8.

3.4.2. Experimental Results of the nRank Algorithm. Because degree centrality can be used as an evaluation index, it is used as a group in the comparative experiment. In addition, the PageRank algorithm and the LeaderRank algorithm are used as two groups in the comparison experiment of the nRank algorithm. The experimental result of the nRank algorithm based on the Karate dataset, when the similarity is 0.5, is shown in Table 3.

According to the experimental results shown in Table 3, for the order of multiple nodes, the results obtained by the three algorithms are the same as the results of the degree centrality evaluation index of nodes. The difference in the arrangement order of node 0 and node 33 is mainly determined by the connection between the two nodes and...
other nodes. Because the nodes connected to node 0 are ranked higher, node 0 is more important than the node 33 for node ranking.

The experimental result of the nRank algorithm based on Dolphins dataset, when the similarity is 0.5, is shown in Table 4.

According to the experimental results shown in Table 4, for the order of most nodes, the results obtained by the nRank algorithm are the same as the results of the degree centrality evaluation index of the nodes. In the nRank algorithm, node 29 is more important than the node 13 because nodes 45 and 51, which are connected to node 29, are ranked higher.

3.4.3. Experimental Results of Software Vulnerability Detection Method Based on Complex Network Community

The spring-shiro-training project is selected as the experimental object, and the precision is used as the evaluation index of the result. Based on the different values of support, self-control experiments will be conducted. The support degrees are 30%, 40%, 50%, 60%, 70%, and 80%, and the result is shown in Table 5.

According to the experimental results shown in Table 5, it can be seen that in the complex network graph, the number of rule subgraphs found and the number of vulnerabilities found are related to different values of support. Based on the different values of support, the relationship between support and the number of rule subgraphs found is given here, as shown in Figure 9.

It can be seen in Figure 9 that the lower the support is, the more rule subgraphs are found; the greater the support is, the less the rule subgraphs are found. This is because the greater the support, the greater the constraints on the graph, which will cause fewer rule subgraphs being mined.
Based on the different values of support, the relationship graph between support and the number of discovered vulnerabilities is given here, as shown in Figure 10. It can be seen in Figure 10, the lower the support is, the more vulnerabilities are found; and the greater the support is, the fewer vulnerabilities are found. This is because the greater the support, the fewer rule subgraphs are mined, so the number of vulnerabilities that can be found will also decrease when matching with the original topological graph.

Based on the different values of support, the relationship graph between support and precision is given here, as shown in Figure 11. It can be seen in Figure 11, the lower the support is, the higher the precision is; and the greater the support is, the lower the precision is. This is because the higher the support, the fewer vulnerabilities can be found. Then the ratio of the number of vulnerabilities found to the total number of vulnerabilities in the software system will decrease.

From the experimental results of the software vulnerability detection method based on the complex network community, it can be concluded that with the increase of support, the number of rule subgraphs mined will decrease. This is because higher support means that the mined graphs are required to appear more frequently. The frequency of subgraphs is required to be higher, and the corresponding rule subgraphs discovered will be reduced. However, it does not mean that less valuable content is mined. By comparing Figures 9 and 10, it can be seen that the decreasing trend in the number of discovered vulnerabilities is much smaller than that of the number of discovered rules. Therefore, although the number of rule subgraphs mined decreases, the most valuable rules are not ignored, and many potential vulnerabilities can still be detected. Although the lower the support is, the higher the precision is, too many subgraphs are generated, and the graph processing process will consume more time. Using such a rule subgraph does not only

Algorithm 1: A vulnerability where conditional judgment is ignored.

Table 1: The experimental result of the algorithm when the threshold is 0.3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>R Algorithm</td>
<td>0.910</td>
<td>0.562</td>
<td>0.695</td>
</tr>
<tr>
<td>Karate</td>
<td>Partition algorithm</td>
<td>0.915</td>
<td>0.856</td>
<td>0.885</td>
</tr>
<tr>
<td>Dolphins</td>
<td>R Algorithm</td>
<td>0.958</td>
<td>0.291</td>
<td>0.446</td>
</tr>
<tr>
<td>Dolphins</td>
<td>Partition algorithm</td>
<td>0.946</td>
<td>0.525</td>
<td>0.675</td>
</tr>
</tbody>
</table>

Figure 7: The community structure of the Karate dataset.
waste time, it cannot effectively prove that there is a potential vulnerability.

Theoretically, the support can be set according to the percentage of the number of vulnerabilities in the program to the total number of program construction, so that the maximum number of frequent subgraphs obtained will not be too many, and the generated rule subgraphs are also reasonable, so that it can reduce the possibility of false detection and improve efficiency. However, there is no scenario in which the distribution of software vulnerabilities can be predicted in advance, so it is still necessary to analyze the situation according to the specific situation.

Table 2: The experimental result of the algorithm when the threshold is 0.32.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>R Algorithm</td>
<td>0.914</td>
<td>0.568</td>
<td>0.701</td>
</tr>
<tr>
<td>Karate</td>
<td>Partition algorithm</td>
<td>0.922</td>
<td>0.864</td>
<td>0.892</td>
</tr>
<tr>
<td>Dolphins</td>
<td>R Algorithm</td>
<td>0.963</td>
<td>0.296</td>
<td>0.453</td>
</tr>
<tr>
<td>Dolphins</td>
<td>Partition algorithm</td>
<td>0.952</td>
<td>0.532</td>
<td>0.683</td>
</tr>
</tbody>
</table>

Table 3: The order of the nodes in the Karate dataset.

<table>
<thead>
<tr>
<th>Ranking</th>
<th>Degree centrality</th>
<th>PageRank algorithm</th>
<th>LeaderRank algorithm</th>
<th>nRank algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node</td>
<td>DC</td>
<td>Node</td>
<td>PR</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>17</td>
<td>33</td>
<td>0.099</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0.076</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>12</td>
<td>32</td>
<td>0.061</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>0.051</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>0.046</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>6</td>
<td>31</td>
<td>0.032</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0.032</td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>5</td>
<td>23</td>
<td>0.028</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>5</td>
<td>6</td>
<td>0.028</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>0.028</td>
</tr>
</tbody>
</table>
Table 4: The order of the nodes in the Dolphins dataset.

<table>
<thead>
<tr>
<th>Ranking</th>
<th>Degree centrality</th>
<th>PageRank algorithm</th>
<th>LeaderRank algorithm</th>
<th>nRank algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node DC</td>
<td>Node PR</td>
<td>Node LR</td>
<td>Node R</td>
<td>Node PR</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>12</td>
<td>17</td>
<td>0.032</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>11</td>
<td>51</td>
<td>0.031</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>11</td>
<td>14</td>
<td>0.031</td>
</tr>
<tr>
<td>4</td>
<td>51</td>
<td>10</td>
<td>57</td>
<td>0.030</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
<td>10</td>
<td>37</td>
<td>0.029</td>
</tr>
<tr>
<td>6</td>
<td>57</td>
<td>9</td>
<td>45</td>
<td>0.028</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
<td>9</td>
<td>33</td>
<td>0.028</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>9</td>
<td>29</td>
<td>0.026</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>9</td>
<td>13</td>
<td>0.026</td>
</tr>
<tr>
<td>10</td>
<td>40</td>
<td>8</td>
<td>1</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Table 5: The operating result of the software vulnerability detection method based on the complex network community.

<table>
<thead>
<tr>
<th>Support (%)</th>
<th>Number of rule subgraphs found</th>
<th>Number of vulnerabilities found</th>
<th>Precision (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>95</td>
<td>135</td>
<td>85.4</td>
</tr>
<tr>
<td>40</td>
<td>87</td>
<td>123</td>
<td>77.8</td>
</tr>
<tr>
<td>50</td>
<td>78</td>
<td>119</td>
<td>75.3</td>
</tr>
<tr>
<td>60</td>
<td>72</td>
<td>112</td>
<td>70.9</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
<td>108</td>
<td>68.4</td>
</tr>
<tr>
<td>80</td>
<td>67</td>
<td>105</td>
<td>66.5</td>
</tr>
</tbody>
</table>

Figure 9: The relationship between support and the number of rule subgraphs.

Figure 10: The relationship between support and the number of discovered vulnerabilities.
4. Conclusions

The vulnerabilities in software systems may not only endanger the functions and performance of the system, but may also leak personal privacy and corporate secrets, causing economic losses. This article proposes a software vulnerability detection method based on complex network community, which uses static detection technology, community structure, node ranking algorithm, and several evaluation indexes and uses the characteristics of the community and the graph matching rules to find similar but different graphs. In this way, the software can be judged whether the conditional judgment is ignored and the corresponding detection of software system vulnerabilities is realized.

The experimental results show that the proposed community partition algorithm and the nRank algorithm have good precision. The vulnerability detection method based on the complex network community can effectively detect the vulnerability that the condition judgment is ignored, which provides a new system vulnerability method for security measurement.

This article mainly detects the vulnerabilities caused by the neglect of conditional judgment, and the types of vulnerabilities can be further expanded in the future.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this study.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. U1636115).

References


