Graph Structure-based Clustering Algorithm for Android Third-party Libraries

Shuyuan Liu and Gang Gan
School of Cyberspace Security, Chengdu University of Information Technology, Chengdu 610225, China. Email: test_me@cuit.edu.cn

Abstract. With the rapid development of the mobile market, the development of multi-functional applications is more efficient due to the rich functions provided by third-party libraries, so it is widely integrated into Android applications. Studies of Android third-party library security, such as hole digging, permissions, separation mechanism, the clone application test and safety test have different requirements to the third party libraries test accuracy and test center of gravity, making the Android third party libraries become a research hotspot. Besides, the detection algorithm of clustering algorithm is particularly important, therefore this article mainly research the Android third-party libraries clustering algorithm. This paper starts with the API call graph of the Android third-party library and combines the graph neural network GAT to design the similarity calculation and library clustering model of the Android third-party library. Firstly, the reverse tool was used to extract the API call diagram of the third-party Android library, and then the third-party Android library instance diagram was built based on the package dependency. Key API functions were selected to normalize the third-party Android library instance diagram, and then GAT and CNN were used as the similarity calculation model of the third-party Android library instance diagram to calculate the similarity. Finally, DBSCAN clustering algorithm is used to cluster the Android third-party library instance graph. Experimental results show that the method proposed in this paper can achieve 93% clustering accuracy and effectively cluster Android third-party libraries.

1. Introduction

In order to adapt to the growing market demand for multi-functional applications and to speed up the development process of mobile applications, packaged third-party libraries are favored by developers because of the rich services they can provide. However, since third-party libraries are developed by other outsiders, and application developers have little knowledge of their internal structure, vulnerabilities in the included libraries can greatly reduce the security of the application. Therefore, in recent years, Android third-party library security research has gradually become a research focus.

At present, the security research of the third-party Android library mainly includes the exploitation of the vulnerability of the third-party library [1, 2], the permission separation mechanism [3], the detection of clone application [4], and the disclosure of privacy information [5]. For the research of third-party library vulnerability mining, it is necessary to detect the specific entry method of the third-party library called by the core code. For the research of permission separation mechanism, cloning application detection and privacy information leakage, it is necessary to detect the complete boundary of the third-party library code. Therefore, the research on vulnerability mining of Android third-party library should be based on the detection of third-party library code. The focus and accuracy of detection greatly affect the above research results.
Mainstream Android third-party library detection is divided into two steps: third-party library clustering and third-party library detection algorithm. Among them, the results of the third-party library clustering algorithm will greatly affect the subsequent detection accuracy, so it is the most important. Therefore, this paper mainly studies the third-party library clustering algorithm.

The current Android Clustering algorithm, a third-party library PiggyApp [6] and AdDetect [7] using Hierarchical Clustering (d), Hierarchical Agglomerative Clustering) thoughts, system structure based on reverse engineering information, by building the Package Dependency Graph (PDG, Package Dependency Graph) to the boundary of the third-party libraries for identification. Since hierarchical clustering is a tree structure that cannot represent multiple dependencies, and this method only builds the clustering tree based on the inclusion relationship in the code file directory, it will give the package too high a weight, thus interfering with the recognition of different third-party libraries with the same prefix name. To solve this problem, Wukong [8] and LibRadar [9] only cluster the API frequency and type within the package, but because the dependency of package level is not considered at all, the identified code is too scattered to fully identify the boundary of the third-party library. LibD [10] clusters the set of homologous packages with the same root node (there is an inclusion or inheritance relationship between packages) according to the function call relationship within each package. However, due to only considering the homologous structure between packages, it is impossible to identify the homologous package sets of different root nodes of the same third-party library whose function call relationship is not obvious.

2. Clustering Model of Android Third-party Library Based on Graph Structure

2.1. Android Third-party Library Instance Diagram

The overall framework of the Android third-party library clustering algorithm proposed in this paper is shown in Figure 1. First, the DEX file was analyzed by the reverse analysis method and its package dependency graph and function code characteristics were extracted to construct the property graph, and then the property graph was processed to match the input of the heterogeneous clustering algorithm.

Figure 1. Overall Frame Diagram
2.1.1. API Call Diagram Constructs. The API call graph, that is, the code of the third party library of Android, USES a graph structure to represent the sequence of the execution of the API call during the execution process. In this way, the execution status and order of the API in the code package of the third party library of Android as well as the overall characteristic information of the API can be fully preserved. After decompression of Android APK file, DEX binary code file is obtained, and then converted into API call graph. The graph structure is adopted to represent the characteristics of Android third-party library code. The process is shown in figure 2.

In this article, the WALA script is used to automatically extract the API call graph of the DEX file, and the reverse information of the Android DEX file can be obtained through apktool. Recursive method is adopted to traverse function calls of DEX file, and skip instructions including goto, package-switch, etc., and function call instructions including invoke-virtual, invoke-super, invoke-direct, etc. Are selected. Remove developer custom functions and Android library source functions from Android programs based on the androidmanifest.xml file in the APK installation package. If the target is an Android third-party library function, go inside the function, continue to scan its internal instructions, and filter the API. After the filtering, according to the sequence of API execution, connect the apis in different functions and different basic blocks to establish the API call graph of the Android third-party library.

2.1.2. Android Third-party Library Instance Diagram Construction. In this paper, the API call diagram containing package dependency diagram is used to study the object, where package dependency refers to the dependency relationship between each package, and the sub-java packages contained in the same Java package will be regarded as independent packages. As an edge between nodes, the dependency relationships in this paper involve the following four types: function call relationships, interface implementation relationships, field reference relationships, and reflection call relationships. Among them, the function call relation is the most common, because the subclass must call the superclass constructor, so the function call relation contains the inheritance relation between classes. The second is the interface implementation relationship. An interface is a keyword representing an abstract set of methods in the Java language. A class implements an interface method via the implements keyword. The reflection invocation relationship is then included in the Dalvik annotation.

After the Package dependency diagram is constructed, the code characteristics of the Java Package are added as attributes to supplement the API call diagram. The code features include the dependencies among the four packages mentioned above. The functions in each package that have dependencies are connected with their corresponding package dependencies to establish the overall API call graph.

2.2. Key Node Selection
There is a certain gap in the scale of the instance graphs of different Android third-party libraries. In order to apply the input of the similarity calculation model of the graph neural network used in this paper, its scale needs to be standardized, that is, the number of nodes of the instance graph needs to be fixed. The similarity calculation model includes the full connection layer, and the number of neurons in the full connection layer is fixed and corresponding to the dimension of the input vector, so the key API selection should be carried out. Traditional key API selection algorithms are used to express the importance of nodes by the external attributes of nodes, such as node degree and proximity degree. However, in the API call diagram in this paper, each node represents different apis with different meanings, so it is not easy to select key nodes by traditional methods.
For the calculation of the importance of nodes in the instance diagram of Android third-party libraries, in addition to calculating the importance of individual nodes, the interaction between nodes should also be considered. There are five types of dependency edges between nodes in this paper, and different edges have different degrees of importance, so they should have different weights. Therefore, this paper USES Textrank [11] algorithm for reference to calculate the importance degree of nodes in the instance diagram of the third-party Android library. At the same time, it classifies the API, sets different levels for different APIs, and then calculates the importance degree of each node in the diagram to select the key nodes.

The basic idea of Textrank algorithm is that text is a network made up of words or a graph made up of words, and semantic relationships between words constitute the edges of the graph. The more important a word is in the graph, the more likely it is to become a keyword. As the relationship between words in the text is linked to each other, the score calculation of Textrank is an iterative process, finally ranking the importance of words according to the ranking of the score. The calculation formula of Textrank node score can be judged by equation (1).

\[
TR(V_j) = (1-d) + d \sum_{V_i} W_{ji}^{TR}TR(V_i)
\] (1)

In the equation, \(TR(V_j)\) is the Textrank value of the word \(V_j\), and for each node \(V_i\), \(W_{ji}\) represents the weight of the edge between \(V_i\) and \(V_j\). \(d\) is the probability of a word appearing randomly on the page. The initial value of \(d\) is usually set to 0.85.

The Textrank algorithm can be used to calculate the importance of each node in the instance diagram of an Android third-party library, but there are some differences between this approach and the relationships between words in the text. Android third-party library instance graph API nodes exist in different package bag, rely on the mapping relationship between the mapping relationship between corresponds to the different weights, the weights of relationships to the third party libraries of Android code dependencies contribution degree are different, so in the process of node calculation for API calls the picture, need to transfer the contribution of API itself into consideration.

In order to calculate the contribution of different package dependencies in the instance diagram of the Android third-party library, it is necessary to consider whether each package appears separately in the Android third-party library. With the help of the Software Development Kit that comes with Android, the toolkit integrates more than 80 third-party library toolkits, analyzes the samples of the Android third-party library with apktool, and is able to provide package dependency evaluation. By randomly selecting 7830 third-party Android library samples in the past two years, we counted the frequency of package dependencies in the samples and calculated the tf-idf (term frequency-inverse document frequency) value of each package dependency. The calculation formula of tf-idf node score can be judged by equation (2).

\[
tf-idf_{i,j} = \frac{n_{i,j} \log \frac{|N|}{|\{j : t_i \in d_j\}| + 1}}{\sum_{i} n_{i,j}}
\] (2)

In the equation’s each API node, \(tf-idf_{i,j}\) is the TF-IDF value between each API node, \(n_{i,j}\) is the total number of samples containing API call \(i\) in category \(j\), and \(|N|\) is the total sample number. The denominator is the total number of samples in category \(j\), and \(|\{j : t_i \in d_j\}|\) is the number of samples containing API call \(i\).

In summary, the TF-IDF is introduced in the process of calculating the node contribution, and the value can be judged by equation (3).

\[
tR(v_j) = \frac{tf-idf_{i,j}}{N} + tf-idf_{i,j} \cdot \sum_{d_j} \frac{R(v_j)}{L(v_j)}
\] (3)
In the equation, $R(v_i)$ is the degree of contribution of node $V_i$. Because only the contribution of the API to the Android third-party library sample is considered, the TF-DF value is the TF-DF value of the Android third-party library sample category. The number of nodes $V_j$ calls nodes. According to the recursive calculation of equation (3), when the result tends to be stable, the contribution degree of the node is obtained.

### 2.3. Similarity Calculation

#### 2.3.1. Graph Embedding Algorithm

In this paper, GAT [12] and CNN are used to construct the similarity calculation model as shown in figure 3, which is a Siamese double-path neural network and can be divided into two stages: graph embedding and similarity calculation. Graph embedding is to convert the context information of the vertices of a graph into a low-dimensional, dense representation vector, which is used to represent the structural characteristics of the whole graph. The graph embedding algorithm adopted in this paper uses the double-path GAT model to obtain the graph embedding. A layer of GAT allows the graph structure information to be transferred between first-order nodes, but for the instance graph of Android third-party library, there is no correlation between any two graph structures. Therefore, the double-path GAT model is used in this paper to correlate the instance graphs of two Android third-party libraries, so that the similarity between the two graphs can be calculated.

In the embedding phase of the graph, first of all, nodes are embedded in the form of adjacency matrix and the output embedding matrix is $V = VN \times S$, where $N$ is the number of nodes and the number of rows of the node embedding matrix. $S$ is the dimension of the embedded matrix. As the input of GAT algorithm, the embedded matrix is divided into two stages: attention coefficient calculation stage and weighted sum stage.

In the attention coefficient calculation phase, for each vertex in the graph structure data, the similarity coefficient between its neighbors and itself is calculated one by one. And the value of attention coefficient can be judged by equation (4).

$$e_{ij} = a([W_{vi} \| W_{vj}]), j \in N_i \tag{4}$$

In the equation, $W$ is a shared parameter, and its linear mapping adds dimension to the vertex features, which is a common feature enhancement method; $[W_{vi} \| W_{vj}]$ stitches the transformed features of vertices $i, j$; finally a $([W_{vi} \| W_{vj}])$ maps the high-dimensional features after stitching to a real number. Then use the softmax function to normalize the attention coefficient. Note the coefficient normalization can be judged by equation (5).

$$\theta_{ij} = \frac{\exp(\text{Leaky ReLU}(e_{ij}))}{\sum_{k \in N_i} \exp(\text{Leaky ReLU}(e_{ik}))} \tag{5}$$

The activation function can be calculated by equation (6).

$$y_{ij} = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0 \end{cases} \tag{6}$$

In the weighted summation stage, the features are weighted and summed according to the calculated attention coefficient. Feature Weighted Sum Value can be calculated by equation (7).

$$V_i' = \sigma(\sum_{j \in N_i} \theta_{ij} W_{ij}) \tag{7}$$

In the equation, $V_i'$ is the new feature of each vertex $i$ output by GAT.
2.3.2. Graph Embedding Algorithm. After the graph embedding phase, the graph structure is represented as a low-dimensional vector. As the input of the CNN layer, the CNN includes a pooling layer and a convolution layer, and the pooling layer selects pooling. Then calculate the matrix similarity through similarity. Fully connected and hidden layers perform feature integration and non-linear transformation. The function outputs the similarity between the two inputs. Compared with ordinary neural networks, the input of the dual GAT model is a sample pair instead of a single sample. The specific function used is the identity function, and the output is the similarity value used for clustering. The value of softmaxloss can be calculated by Equation (8).

\[ L(x_i) = -\log f_i(x_i) = -\log \frac{e^{x_i}}{\sum_j e^{x_j}} \]  

In the equation, \( y = \{y_0, y_1, y_2, \ldots, y_n\}, \ y_i \in \{0,1\} \) is the category description of \( x_i \). This function is used to calculate the error between the real category and the training category.

2.4. DBSCAN Clustering

DBSCAN (Density-Based Spatial Clustering of Application with Noise) clustering algorithm is a density clustering algorithm. Compared with the K-means clustering method, this method does not need to implement a specified k value, that is, the number of clusters. It is suitable for clusters with irregular shapes and specific evolution trends, and for irregular clusters formed by the uncertain number of packages in the Android third-party library code.

**Algorithm 1 Density-Based Spatial Clustering of Application with Noise, DBSCAN**

Input: a graph \( G(V, E), MinPts \)

Output: a new graph \( G-cluster \)

1: Mark all objects as unvisited
2: Do
3: Randomly select an unvisited object \( v \)
4: Mark \( v \) as visited
5: If \( v \)'s \( v \in U(\sigma) \) has at least \( MinPts \) objects
6: Create a new cluster \( C \) and add \( v \) to \( C \)
7: The set of objects in \( \delta \)-domain where \( N \) is \( v \)
8: for each in \( N \)
9: if \( v \) is unvisited
10: Mark \( v \) as visited
11: if \( v \)'s \( \delta \)-domain has at least \( MinPts \) objects, add those objects to \( N \)
12: if \( v \)' is not already a member of any cluster, add \( v \)' to \( C \)
13: END for
14: Else drop \( v \)
15: Return \( G-cluster \)

Algorithm 1 gives the pseudo code of the DBSCAN clustering algorithm, where the input of the algorithm is the Android third-party library instance graph, and is the number of objects. First mark all nodes in the Android third-party library instance graph as unvisited (line 1), then traverse the graph to mark all objects in the node's domain as a cluster (Algorithm 2-13), and then mark non-node The nodes in the domain are all discarded as noise, and finally the Android third-party library instance graph is output.
3. Experimental Design

3.1. Concepts of Fuzzy Analytic Hierarchy Process (FAHP)

This article uses the Google Play data set provided by Androzoo. The final data set contains a total of 2195863 different applications with SHA256, including 2743843 applications with different package names. The average size of applications in the data set is 4.3M bytes, totaling 16.5TB.

In the Android third-party library clustering algorithm, accuracy and recall are both very important indicators. The recall rate indicates the number of third-party libraries that can be correctly identified; the accuracy rate indicates the importance of the detection results. Therefore, we use the adjusted Rand index and clustering accuracy as the evaluation index of the clustering effect of the third-party library of Android.

ARI is a modification of RI, the RI calculation’s equation is as shown in equation (9).

\[ RI = \frac{\alpha + \beta}{c_2^n} \]  

In the equation, \( n \) indicates the number of elements in a given set, and there are \( c_2^n \) set pairs in the set; \( \alpha \) indicates the number of pairs of homogeneous elements that are classified into the same cluster; \( \beta \) represents different types of elements classified into different clusters. However, the clustering algorithm proposed in this article hopes that when the clustering result is a random value, the index will infinitely approach 0, and the index cannot meet this situation, so it is used in this article, and the specific definition is as shown in equation (10).

\[ ARI = \frac{RI - E(RI)}{\max(RI) - E(RI)} \]  

The \( ARI \) value range is \([-1, 1]\), the larger the value, the better the effect.

4. Experimental Results

In order to get the best Rand index for the experiment, set the Eps range from 0.1 to 0.95, with an interval of 0.02. The previous experiments show that the minPts parameter has a small effect on the experimental results, and the selected minPts value is 9. The distribution diagram is shown in Figure 3, where the abscissa is the Eps value and the ordinate is the adjusted Rand index.

Compared with the method proposed in this paper (hereinafter referred to as GAT) and LibD, the experimental results show that the best Eps values are 0.82 and 0.8, respectively.

![ARI Distribution Chart](image)

Figure 3. ARI Distribution Chart

At the same time, because the graph neural network algorithm is used to greatly improve the algorithm efficiency, the processing time of LibD and GAT algorithms is shown in Figure 4.
5. Conclusion

Since the previous clustering algorithm only uses code features and does not combine the dependency structure relationship between packages for clustering, a graph structure-based Android third-party library clustering algorithm is proposed, which can be fully expressed. This paper first proposes a clustering algorithm for Android third-party library based on graph structure. This algorithm uses graph neural network and clustering algorithm to process graph structure data, which provides a new idea for cluster analysis of third-party library in Android. Secondly, the API call graph package extracted by reverse analysis was combined with the package dependency graph to transform, and the Android third-party library instance graph was defined. Finally, the application of graph convolutional neural network to the cluster analysis of Android third-party libraries is proposed, and the similarity calculation model of Android third-party library instance graphs is designed. At the same time, the Android third-party libraries are clustered. A large number of experiments were performed on the public Android third-party library data set, and the accuracy of the proposed method reached 93%.

Future work can be devoted to using new methods, relying on source code and package structure dependencies to the maximum to identify third-party library instances, rather than relying on multiple hashes of feature value vectors. At the same time, in order to avoid false positives caused by incomplete prior knowledge, automatic feature extraction can be used to extract feature vectors from source code to reduce manual intervention and improve efficiency while improving efficiency.

6. References

[1] Bhorskar R, Han S, Jeon J, et al. Brahmastra: driving apps to test the security of third-party components[C]//Proceedings of the 23rd USENIX conference on Security Symposium. USENIX Association, 2014: 1021-1036.

[2] Zhang Chao. Research on Android-based third-party class library vulnerability mining technology [D]. Tianjin University of Technology

[3] Liu B, Liu B, Jin HX, View R. Efficient privilege de-escalation for ad libraries in mobile apps. In: Proc. of the 13th Int l Conf. on Mobile Systems, Applications, and Services (MobiSys 2015). New York: ACM Press, 2015. 89 103. [doi: 10.1145/2742647. 2742668

[4] Wang HY, Guo Y, Ma ZA, Chen XQ. WuKong: A scalable and accurate two-phase approach to Android app clone detection. In: Proc. of the ACM Int l Symp. on Software Testing and Analysis. New York: ACM Press, 2015. 71 82. [doi: 10.1145/2771783. 2771795

[5] Jung-Hyun Lee, Woo-Jong Ryu, Kang-Min Kim, and SangKeun Lee. 2019. MoCA: a novel privacy-preserving contextual advertising platform on mobile devices. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC ’19). ACM, New York, NY, USA, 1208-1215. DOI: https://doi.org/10.1145/3297280.3297399

[6] Wu Z, Zhou Y, Grace M, et al. Fast, scalable detection of Piggybacked mobile applications[C]// Acm Conference on Data & Application Security & Privacy. 2013.

[7] Narayanan A, Chen L, Chan C K. AdDetect: Automated detection of Android ad libraries using semantic analysis[C]// 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP). IEEE, 2014.

[8] Wang H, Guo Y, Ma Z, et al. WuKong: A Scalable and Accurate Two-Phase Approach to Android App Clone Detection[C]// the 2015 International Symposium. ACM, 2015.
[9] Ma Z, Wang H, Guo Y, et al. LibRadar: fast and accurate detection of third-party libraries in Android apps [J]. 2016.

[10] Li M, Wang W, Wang P, et al. LibD: Scalable and Precise Third-Party Library Detection in Android Markets[C]// 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE Computer Society, 2017.

[11] Barrios F, López F, Argerich L, et al. Variations of the similarity function of textrank for automated summarization [J]. arXiv preprint arXiv:1602.03606, 2016.

[12] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks [J]. arXiv preprint arXiv:1710.10903, 2017.