ATLANTIS PRESS
Advances in Computer Science Research, volume 88
International Conference on Computer, Network, Communication and Information Systems (CNCI 2019)

An Android Malware Detection Approach Based on Weisfeiler-Lehman Kernel

Jie Ling[a] and Fangye Chen[b]
School of Computer Guangdong University of Technology Guangzhou, China
[a]jling@gdut.edu.cn, [b]lingnancfy3090@gmail.com

Abstract. The explosive growth of Android malware has caused great harm to users' lives and work. In this work, we propose an Android malware detection approach based on Weisfeiler-Lehman (WL) kernel, which converts malware detection problem into similarity analysis problem of function call graph, and introduce contextual information of function call process to enhance nodes label of function call graph. The node label enables the feature space to contain both the structural information and the contextual information of the graph. The similarity of the function call graph is calculated by the Weisfeiler-Lehman graph kernel algorithm to detect the Android malware. Our experimental results show that the WL kernel method enhanced by contextual information is higher than three state-of-the-art kernel methods of CWLK, NHGK and WLK and two classical detection methods of Drebin and Androguard in precision and recall rate.

Keywords: Malware, graph kernel, call graph, context.

1. Introduction

Android malware has seriously threatened the privacy and property security of mobile users. According to the 360 Internet Security Center's 2017 malware special report [1], the Android platform added 7.573 million new malware, an average of 21,000 per day. Malware is developing rapidly and most of them comes from variants. Researching variant malware detection can better achieve the purpose of Android malware detection.

In order to achieve automated detection of Android malware, as in [2-5], malware can be identify by feature matching, which quickly detects existing malware but does not recognize code-confused variants of malware. In [6], the machine learning method is used to extract the sensitive APIs and permissions in the application to form feature vectors, and the Bayesian network is used for malware detection. This method does not analyze sensitive API calls in a specific context, and there is a high probability of detection errors for variant malware. Some security researchers introduce contextual information to analyze malware. The APPContext[7] detection system uses the soot tool to obtain function call graphs, extracting contextual feature vectors which constructs by activation events and context factors, and then classifiers was used to detect malware.

At present, the most effective means to combat code confusion attacks is use graph similarity analysis technology. The basic principle is to transform the detection problem of Android malware into graph isomorphism. First step is use the open source tool to extract the program representation from the APK file, and then implement the similarity measure of the malware through the graph isomorphism algorithm. Many graph isomorphic algorithms have been proven to be NP-hard problems and cannot solve program representations with large node sizes. Reference [8] combined with dynamic analysis and static analysis, the subgraph matching algorithm is used to measure the similarity between the application framework API and the kernel call sequence in a malware family, and whether the application is malware based on the simple average of the similarity measure. A number of approaches[9-11] are based on the graph embedding algorithm which structured Information of the function call graph is embedded into vector space and machine learning algorithm is used for classification prediction. However, the defect of the graph embedding algorithm is that a large number of graph structure features are lost in the process of transforming the graph into feature vectors, which reduces the precision of detection. Many scholars use the graph kernel to solve the graph isomorphism problem, and propose a series of graph kernel

Copyright © 2019, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

frameworks[12-15] which have been applied in biomedicine, chemistry, social networks and so on. These graph kernel can only obtain the structure information of the graph without including the context information, and cannot be directly used for Android program graph analysis. We used Androguard [16] to extract a contextual information feature vector including activation events and context factors. Such feature vector can be able to use to enhance graph nodes in the computational procedure of WL graph kernel. As a result, the feature space of WL kernel contain both structure information and contextual information which has been proven to improve detection accuracy of android malware. Ultimately, a kernel matrix is calculated by the WL kernel algorithm, each element in the kernel matrix represents the similarity measure of the two graphs. Applying the matrix to the SVM classifier for model training and testing can obtain malware detection results. The detection framework is shown in Fig. 1.

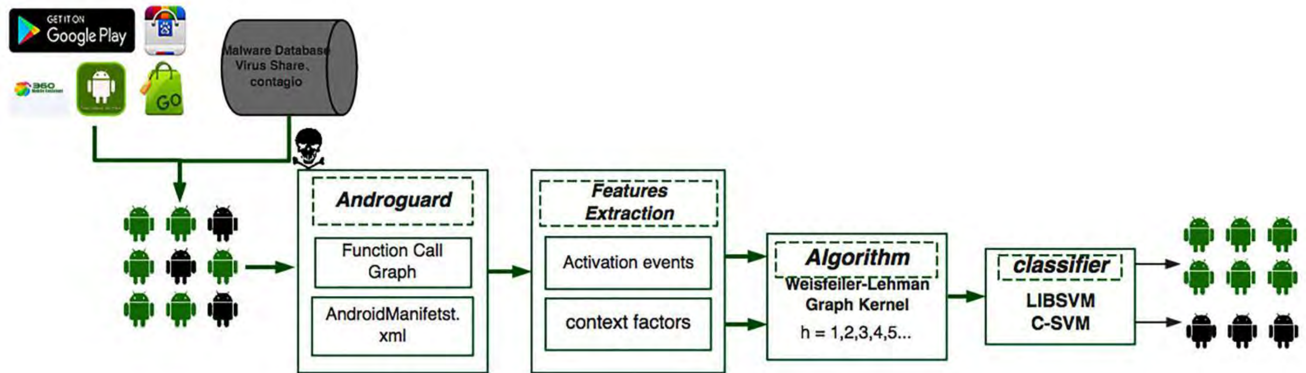

Fig. 1 Android Malware Detection Framework.

2. Introduction of Related Methods

2.1 Activation Events and Context Factors.

The activation events and context factors extraction are based on function call graphs and sensitive APIs. Reference [17] has given a list of all sensitive APIs on the Android platform. We uses Androguard to analyze the APK data to get function list. Comparing the function list with the standard sensitive API list, and then we can locate the sensitive API that the Android application calls. It important to save the function name and entry point of the API for later analysis.

Context information mainly contains two characteristics: activation events and context factors. These two features together determine the calling of sensitive methods. An activation event refers to an external event that drives a sensitive method call. In general, normal software interacts with the user when it call the sensitive methods like sendMessage() and sendDeviceId(). Malware generally calls sensitive methods in the background since the user is unknown. The activation events are divided into two cases: user known and user unknown, which can be represented by 1 and 0. An context factor is a device environment property that controls the execution of a sensitive method. For example, MoonSms, the DroidDream family's malware, drives malicious behavior through changes in cell phone signals; FackBank executes malicious code in the middle of night. The context factors on the Android device include the device current time Tf, the screen state Ss, the signal strength Sg, and so on, which can also be represented by 1 and 0.

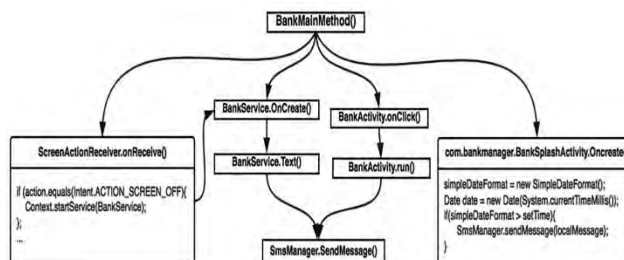

Fig. 2 FakeBank malware code call process.

FakeBank is a malicious application imitated the Korean bank KoreaBank. As shown in Fig. 2, we have found that FakeBank calls the sensitive method SendMessage() in three ways by studying the source code of FakeBank. Firstly, user calls the onClick function by pressing the button to run SendMessage(),which is an activation event known to the user and belongs to normal software behavior. The second way to call that function appears after the user's phone screen is dimmed. FakeBank sets up a ScreenActionReceiver receiver to listen for the "SCREEN_OFF" signal. Once the screen is dark, the BankService service is started to send a text message. This method invokes a sensitive method under the condition of "user unknown" activation event and "SCREEN_OFF" context factors, which is a malicious behavior. The third way is to send a text message directly when the interface starts, which is also a malicious behavior. In our experiment, the activation event recognition algorithm AERA [18] is used to extract the activation event features. activation event set is represented by$\{\xi(n) \mid n \in N\}$, where N is the set of nodes in the function call graph, n is the sensitive method node, $n \in N$. We analyzed the program's binary file, simplifying the function call graph, and extracting the context factors from it. The context factor is represented by $\{\upsilon(n) \mid n \in N\}$, where N is the set of nodes in the function call graph, n is the sensitive method node, $n \in N$.

2.2 Weisfeiler-Lehman Graph Kernel Algorithm.

The Weisfeiler-Lehman kernel is based on the one-dimensional Weisfeiler-Lehman isomorphic test to calculate the similarity of the graph. The basic steps are shown in Algorithm 1. Suppose that a graph structure has given, the idea of the algorithm is to use the ordered label set of neighboring nodes to enhance the node label each time during h iterations. The updated label of each node is equal to the combination of the node label and its neighbor node label, where the node label is in front, and the labels of other neighbor nodes are added in ascending or descending sequence. At the same time, in order to avoid the excessively long label combination affecting the computational complexity, the hash function is used to compress the node label to generate a new node label after each iteration update.

Algorithm 1 one-dimensional WL isomorphic test

```
Input:    program function call G node number vsize
Output:  WL map Gi of the i-th iteration

if(i > 0)
for(i = 0;i<vsize;)
// Calculate the multi-set of adjacent nodes of node v
Mi(v) = L(v);
// Label collection serialization
si(v) = serialize(Mi(v));
// Add the original tag to the tag sequence
si(v) = si(v).append(OriginLabel);
NewLabel = Hash(si(v));
Return Gi
```

We suppose the function call graph is $G = (N, E, \lambda)$, where N is the set of nodes in the function call graph, and each node $n \in N$ represents a function entity. $E \subseteq (N \times N)$ is the edge set corresponding to the function call graph. Each edge $e(n_1, n_2) \in E$ represents the call of the n_1 method to the n_2 method. λ is a set describing whether the node is a sensitive method, and $\ell: N \rightarrow \lambda$ is a node label allocation function. The function call graph that is iterated by the i-dimensional Weisfeiler-Lehman isomorphic test algorithm is represented as$\mathcal{g}_i = (V, E, \lambda_i)$, and the WL graph sequence can be obtained after h iteration:

$$\mathcal{g}_0, \mathcal{g}_1, \dots, \mathcal{g}_h = (V, E, \lambda_0), (V, E, \lambda_1), \dots, (V, E, \lambda_h) \tag{1}$$

If a kernel function k is given and two WL graph sequence of function calls graphs G and $G'$are given, the WL graph kernel function can be obtained:

$$k_{WL}^{(h)}(G, G') = k(\mathcal{g}_0, \mathcal{g}_0{}') + \cdots + k(\mathcal{g}_h, \mathcal{g}_h{}') \tag{2}$$

The final value of $k_{WL}^{(h)}(G, G')$ is the similarity between the two function call graphs G and G'.If we have K samples, through the calculation of the WL graph kernel algorithm, the $K \times K$ diagonal graph kernel matrix can be obtained. Each element in the matrix represents the similarity of G andG'.

2.3 Disadvantages of WL Graph Kernel.

It can be seen from the WL graph kernel function and the one-dimensional isomorphic test algorithm that the WL graph kernel collects neighbor node labels in each iteration, and forms a new node label after serialization and hash calculation. It can be seen intuitively that the WL graph kernel function only describes the structural information of the original graph, and is suitable for application in face image recognition and gene structure graph literacy analysis. For the Android application function call graph, if we want to improve the detection precision of the malware, we should also use the activation event and the context factors to enhance the sensitive node label so that the WL graph obtains the structural information and contextual information of the original graph at the same time.

3. Activation Context Weisfeiler-Lehman Kernel.

The purpose of ACWLK is to enhance the function call graph node label by adding activation events and context factors, so that the function call graph can obtain contextual information and structure information simultaneously after h iteration calculation by one-dimensional isomorphism test algorithm. In our work, a two-step calculation is added in the re-labeling process of each iteration of the WL graph kernel algorithm, which is used to add activation events and context factors to each node. The specific calculation process is shown in Algorithm 2.

Algorithm 2 ACWLK re-labeling algorithm

Input:
1. $G = G_0 = (N, E, \lambda_0, \xi, \nu)$
2. Number of iterations h
Output: $\{g_0, g_1, \dots, g_h\}$ //ACWL Sequence of graphs.

RE-LABEL(G, h)
for $i = 0$ to h **do**
Step 1. for all $n \in N$ and **if** $i = 0$,assign activation events ξ and context factors ν for node *n*.
Step 2. for all $n \in N$ and **if** $i > 0$,construct neighbor nodes set $\mathcal{N}(n)$ and neighbor label set $M_i(n)$.
Step 3. $\lambda(n) \leftarrow \lambda_{i-1}(n) \oplus sort(M_i(n))$,new label is constructed by $M_i(n)$ and last iteration label.
Step 4. Complete the hash calculation for $\lambda(n)$ and prefix the activation events ξ and context factors .
return $\{g_0, g_1, \dots, g_h\}$

We first extract the activation events and context factors in the graph dataset by the methods mentioned above. At the same time, each node in the graph data is serialized, and the obtained activation event and context factor are added to the corresponding node label. Finally, multiple iterations of the algorithm are performed. Therefore, compared with the WLK algorithm, the ACWLK algorithm only improves on the node re-label process. The other steps are consistent, including the hash function used in one-dimensional Weisfeiler-Lehman isomorphic test.

As Algorithm 2, in the first iteration ($i = 0$), it is not necessary to acquire the neighbor node label set, and only the activation event ξ and the context factor ν need to be added as prefixes to each sensitive method node label of the function call graph. When$i > 0$, the neighbor nodes of the node $n \in N$ are stored to the set $\mathcal{N}(n)$, and for all the nodes $m \in \mathcal{N}(n)$, the label of the node m is stored to the set $M_i(n)$. Add $\lambda_{i-1}(n)$ (the node label of the last iteration) to the sorted set $M_i(n)$ to form a new node label$\lambda_i(n)$. For the next iteration, it need to re-add the activation event ξ and the context factors υ as prefixes to the new label and do a hash operation on the new label for time complexity of the algorithm. the function call graph applied to ACWLK is assumed to be $FCG = (N, E, \lambda, \xi, \nu)$. It can be derived from the ACWLK algorithm that If given kernel function k and program call graph FCG and FCG', algorithm 2 can be used to obtain ACWL graph sequence $\{g_0, g_1, \dots, g_h\}$ and $\{g_0', g_1', \dots, g_h'\}$, the similarity values of two function call graphs can be calculated by the following formula:

$$k_{ACWL}^{(h)}(FCG, FCG') = k(\mathcal{g}_0, \mathcal{g}_0') + \cdots + k(\mathcal{g}_h, \mathcal{g}_h') \quad (3)$$

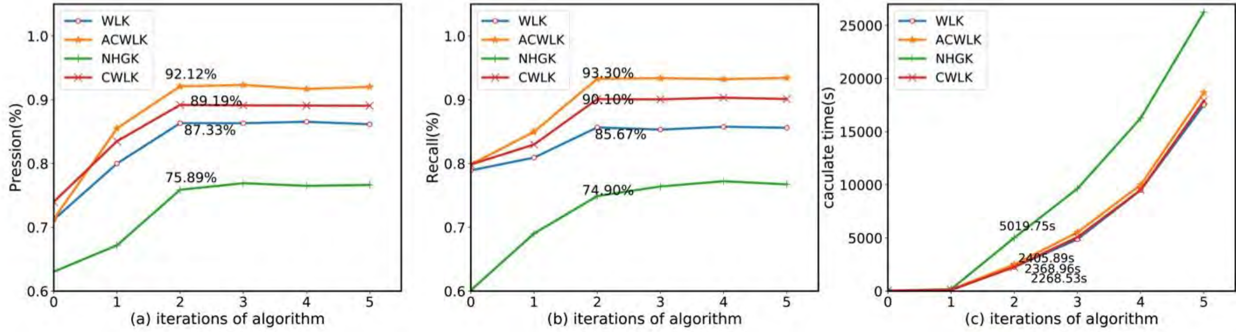

Fig. 3 ACWLK algorithm example

Fig. 3 uses the ACWLK algorithm to analyze two applications, KoreaBank and FakeBank, to illustrate the impact of the algorithm on the precision of malware detection. We can get a partial call graph of SendMessage() by analyzing both App with Androguard. If we analyze the call graph with WL algorithm without contextual information processing, it will get the label graph in step 2. Intuitively, there is no difference between the two call graphs. No matter how many iterations, the WL graph sequences obtained by the two apps make no difference. The malware FakeBank is almost exactly the same as the normal software KoreaBank. If the ACWLK algorithm proposed in this paper is adopted, the two labels "Label_On" and "Screen_Off" will be added in the re-labeling algorithm (FakeBank will perform malicious operation when the user's mobile phone is black, and the context factors is Screen_Off). Only one iteration we can observe the ACWLK map sequence of both app is inconsistent, so that FakeBank can be detected as a malicious application.

The kernel matrix can be obtained by calculation of the ACWL graph kernel algorithm. Assuming K APK samples are given, and finally a $K \times K$ kernel matrix is obtained. Each element in the matrix represents the similarity of K samples to each other. It can be seen from the algorithm process that the algorithm complexity of the ACWL graph kernel algorithm and the algorithm complexity of the WL graph kernel algorithm [12] both can be maintained at $O(K^2h)$ (h is the number of iterations). The relabeling process only consumes a constant level of time complexity.

4. Experimental Results and Analysis

In order to verify the validity of the ACWLK algorithm, ours experiment mainly evaluated the precision the recall. After obtaining the kernel matrix, the C-SVM classifier provided by LIBSVM [19] was used for malware detection classification and a 5-fold cross-check was completed. The malicious samples were mainly from VirusShare [20] and contagio malicious sample website [21]. The benign samples were mainly from Google Play and Baidu Mobile Software Center. The data set was shown in Table 1. 80% of them were used as training sets and 20% were used as test sets.

Table 1 Dataset Composition.

Dataset(APK)	Data Source	Number Of Sample
malware	Virus Share/contagio	2256
benign	Google Play/Baidu	1537

The first step of the experiment compared the results of ACWLK and three state-of-art graph kernel, including CWLK [12], Neighborhood Hash Graph Kernel(NHGK) [13], and Contextual Weisfeiler-Lehman Graph Kernel(CWLK) [14] in the same data set and experimental environment. The results obtained by the four algorithms after 5 iterations were shown in Fig. 4.

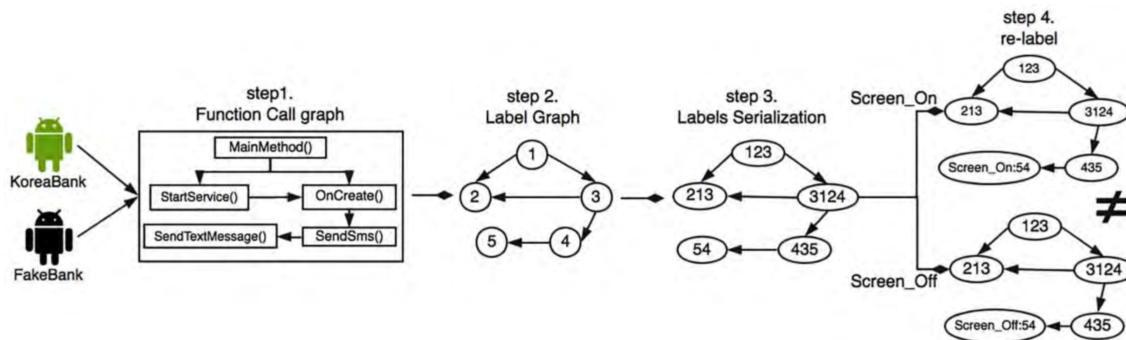

Fig. 4 Comparison of Calculation Results of Different Graph Kernel Algorithms.

As shown in Fig. 4 (a) and Fig. 4 (b), when the number of iterations h>2, the precision and recall rate of the four algorithms do not change much. When h = 0, the discrimination between CWLK and the other three graph kernels is not obvious, but as the number of iterations increases, the two evaluation parameters of ACWLK are significantly higher than the other three graph kernel algorithms. When h=2, the precision and recall rate of ACWLK algorithm are 92.12% and 93.30%, respectively, which is significantly higher than 89.19% and 90.10% of CWLK algorithm. The precision and recall rate of the WLK algorithm are 87.33% and 85.67%, while the NHGK with graph embedding algorithm has only 75.89% precision and 74.90% recall rate. The result of Fig. 4 (c) is the calculation time analysis required for the calculation of 100 samples in the four algorithms. The calculation time for ACWLK is slightly higher than CWLK and WLK, but much lower than NHGK. The CWLK graph kernel algorithm only obtains the activation event without context factors, while WLK and NHGK fail to obtain the contextual information in the function call graph. This experiment shows that adding context labels can surely improve the precision and recall rate for Android malware detection and have little effect on the time complexity of the algorithm.

In the second step of the experiment, we compare the ACWLK algorithm with two Android malware detection methods DREBIN and Androguard that do not use the graph kernel method, and take the ACWLK algorithm's result (iteration number h=2) on the same training set and test set. Comparing three detection methods, we got the results in Table 2.

Table 2 Compare result of defferent detection method.

Detection Method	Number of Total Samples	Number Of Malicious Samples	Precision(%)	Recall(%)
DREBIN	3793	2256	79.50	77.62
Androguard	3793	2256	86.97	83.24
ACWLK	3793	2256	92.12	93.30

It can be seen from the comparison results in Table 2 that the ACWLK using the graph kernel algorithm is higher in precision and recall than the other two detection methods that do not use the graph kernel algorithm. ACWLK can correctly detected 2078 in 2256 malicious samples, indicating that ACWLK based on graph kernel algorithm can effectively improve the precision and recall rate of Android malware detection.

5. Conclusion

We analyze the contextual information and function call graph characteristics of Android malware sensitive API, and propose a Android malware detection method based on Weisfeiler-Lehman kernel algorithm. The function node information is enhanced by activation event and context factors, and the Android malware detection problem is transformed into the graph isomorphism problem. At the end, the graph kernel is used to calculate the graph isomorphism. In our work, the effectiveness of ACWLK algorithm is verified by two-part experiments. In the first place, the ACWLK algorithm was compared with CWLK, WLK and NHGK. It shows that increasing contextual information makes the WL graph kernel more suitable for android malware analysis. In the second place, we compared the ACWLK algorithm with two detection methods of Drebin and Androguard, the ACWLK detection method based on graph kernel is better than the

method based on machine learning and feature matching. Experiments show that the ACWLK algorithm enhanced by contextual information can improve the precision and recall rate of Android malware detection.

Acknowledgements

This work is supported by the science and technology project of Guangdong Province (No.2017B090906003) and the project of Guangzhou Science and Technology(No.201802010043、201807010058).

References

[1]. 2017 Malware Special Report. URL. https://www.freebuf.com/articles/paper/164398.html.

[2]. Li Y, Liu F, Du Z, et al. A Simhash-Based Integrative Features Extraction Algorithm for Malware Detection[J]. Algorithms, 2018, 11(8): 124.

[3]. Faruki P, Ganmoor V, Laxmi V, et al. AndroSimilar: robust statistical feature signature for Android malware detection[C]//Proceedings of the 6th International Conference on Security of Information and Networks. ACM, 2013: 152-159.

[4]. Sheen S, Anitha R, Natarajan V. Android basedmalware detection using a multifeature collaborative decision fusion approach[J]. Neurocomputing, 2015, 151: 905-912.

[5]. Su X, Zhang D, Li W, et al. A deep learning approach to android malware feature learning and detection[C]//Trustcom/BigDataSE/I SPA, 2016 IEEE. IEEE, 2016: 244-251.

[6]. Yerima S Y , Sezer S , Mcwilliams G , et al. ANew Android Malware Detection Approach Using Bayesian Classification[C]// 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2013.

[7]. Yang W, Xiao X, Andow B, et al. Appcontext: Differentiating malicious and benign mobile appbehaviors using context[C]//Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015: 303-313.

[8]. Peng Chen, Rongcai Zhao, Zheng Shan, Jin Han,Meng Wei.A similarity detection of Android malicious code behavior based on dynamic and staticcombination[J].Application Research of Computers,2018,35(05):1534-1539.

[9]. Gascon H, Yamaguchi F, Arp D, et al. Structural detection of android malware using embedded call graphs[C]//Proceedings of the 2013 ACMworkshop on Artificial intelligence and security. ACM, 2013: 45-54.

[10]. Kaspar R, Horst B. Graph classification and clustering based on vector space embedding[M]. World Scientific, 2010.

[11]. Fu Y, Ma Y. Graph embedding for pattern analysis[M]. Springer Science & Business Media, 2012.

[12]. Shervashidze N, Schweitzer P, Leeuwen E J, et al. Weisfeiler-lehman graph kernels[J]. Journal of Machine Learning Research, 2011, 12(Sep): 2539-2561.

[13]. Hido S, Kashima H. A linear-time graph kernel[C]//Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on. IEEE, 2009: 179-188.

[14]. Narayanan, Annamalai, et al."Contextual Weisfeiler-Lehman Graph Kernel For Malware Detection." The 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, 2016.

[15]. Navarin N, Sperduti A, Tesselli R. Extending local features with contextual information in graph kernels[C]//International Conference on NeuralInformation Processing. Springer, Cham, 2015: 271-279.

[16]. Costa F, Grave K D. Fast neighborhood subgraph pairwise distance kernel[C]//Proceedings of the 27th International Conference on International Conference on Machine Learning. Omnipress, 2010: 255-262.

[17]. Desnos A. Androguard: Reverse engineering, malware and goodware analysis of android applications[J]. URL code. google. com/p/androguard, 2013: 153.

[18]. https://github.com/secure-software-engineering/FlowDroid/blob/master/soot-infoflow-android/SourcesAndSinks.txt

[19]. Zhengjun Lu, Yong Fang, Liang Liu, Wenjie Zhang, Zuo Zheng.A Method for Android Malicious Behavior Detection Based on Contextual Information[J].Computer Engineering,2018,44(07):150-155.

[20]. Chang and C.-J. Lin. LIBSVM: A Library For Support Vector Machines, 2001. https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[21]. Virus Share malware dataset. URL. http://virusshare.com.

[22]. contagio database. URL. http://contagiominidump.blogspot.com/.