MULTIPLE ANDROID PACKAGE FILES EXTRACTOR IN MINING REQUEST PERMISSIONS AND API CALLS

A. Aminordin[1], M.A. Faizal[2], Y. Robiah[2], A. Mukhlis[3] and F. Arif[4]

[1]Faculty of Computer and Mathematical Sciences,
Universiti Teknologi Mara Melaka, Jalan Lendu, 78000 Alor Gajah, Melaka, Malaysia.

[2]Faculty of Information and Communications Technology,
Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia.

[3]Rant.ai Network,
Damansara Foresta, Bandar Sri Damansara,
52200, Kuala Lumpur, Malaysia.

[4]Department of Industrial Engineering,
Institut Teknologi Nasional (Itenas), Bandung, Indonesia.

Corresponding Author's Email: [1]azmi1107@melaka.uitm.edu.my

Article History: Received 6 April 2018; Revised 26 July 2018; Accepted 4 October 2018

ABSTRACT: Android smartphone has the highest demand in the world due to the ability of the devices and the open source software concept. Numbers of Android applications are increasing as to fulfill users and businesses' needs. Not only Android gains huge business return but its applications has also become the target of attackers. One of the approaches to investigate and detect malware is through a reverse engineering technique where the profile parameters are extracted. The process of reversing Android execute file (.apk) individually takes a long time. Other than having used several tools, the approach leaves open the possibility of misconduct during the mining of necessary source codes. Therefore, an Android permissions and Application Programming Interface (API) calls extractor tool were developed for Android mobile devices apps. This tool had the capability to record all request permissions and required API calls inside the AndroidManifest.xml and classes.dex made to App executable file. In addition, the automatic feature of the tool allowed for the recording of the permission and API calls more than one Android Package Kit (APK) files at a time. MAPE (Multiple Android Package Extractor) was developed using Node.js. Currently, researchers either disclose mining techniques or use existing tools manually. MAPE used a sequential search in Depth First Search (DFS) technique to accomplish the

operation. This tool can shorten the researchers' processing time on retrieving request permissions and targeting API calls. The output produced by MAPE can be used for several purposes such as Apps categorization and malware detection.

KEYWORDS: *Android; Permissions; API Calls; Depth First Search; Sequential Search*

1.0 INTRODUCTION

Rapid development of Android Apps has triggered serious security concerns to overcome the presence of malware. Malware creator does not only target the business, finance and social apps to inject the vulnerable codes but also to vary the apps category. For example, plenty of engineering apps can be downloaded through either official or third party store. Apps such as *Axon Calc, Heat Transfer Calculator* and *Engineering Cookbook* are among the top 10 Android apps for engineers commonly used by college students and practitioners. Those others apps might be infected by malware as they can be retrieved through a third party app store [1, 2-3]. When a file is downloaded, it does not actually deploy the app that user asks for but a fake Flash player which is being used to spread a malware. According to a study [4], most of the repackaged applications in third party app store are malware applications.

Various methods have been proposed by many researchers and organizations in addressing malware issues. Static and dynamic analyses as well as a combination of both (hybrid) [5] are often used for classifying benign and malware Apps. *Static analysis* is where the source codes is analysed without running the application. It is an inexpensive way to find malicious activities in code segments and consume fewer resources. Failure at different code obfuscation is the main drawback of this technique. *Dynamic analysis* is also known as behavioural based analysis where the technique is executed within a sandbox. Lastly, *hybrid analysis* integrates the two approaches. Features used to analyse malware for those techniques are clearly explained [6].

Furthermore, static analysis considers features such as permission, intents, API calls, broadcast receiver and string written in the source codes while dynamic analysis investigates system call, power consumption, network traffic and user interactions [6]. In static analysis, permissions [7] and API call [3–6] are important features used to detect malware. Permission is the security for the system and users before the

apps can use certain system data and features. All request permissions are declared inside AndroidManifest.xml file. On the other hand, API calls are the set of subroutine definitions, protocols and tools for building application software. In the Android environment, all APIs are coded into their specific classes and grouped into one file named classes.dex.

Android APK is similar to ".exe" file in the Windows environment. Several files and folders are compiled and packaged to merge as a single file with the ".apk" extension. This file can easily be decompressed using Unzip tools, but it will be returned in an unreadable format. The important files are (i) AndroidManifest.xml which contains essential information about the app where the system must have the access before it can run any of the app codes. (ii) Classes.dex contains code that is written in java and then compiled to class files before these files are cross compiled to Dalvik VM format. (iii) Resources.arsc: arsc stands for application resource. This file is kept in the compiled resources (strings, images, & data) in a binary format.

Android lists the package index for every version of API level in its developer page. The combination of package index with specific classes also known as API call is used to activate the Apps. In addition, API calls provide means for Apps to interact with the devices where the static API calls give information on their runtime activities[1]. Many researchers on Android Apps categorization [8-9] and malware detection [7, 10–20] choose API calls as one of the features in their studies. Recent studies [15, 21] considered API in their study of Android malware detection, thus, showing that this feature is still relevant to be analyzed.

ApkTool is a tool for reverse engineering that decodes resources to their nearly original forms and rebuild them after some modifications. This tool is similar to any unzipped utilities tool that will unzip the APK file but this tool allows the transfer of smali files to smali folder [26]. This command line tool allows users to view the request permissions inside the AndroidManifest.xml.

Table 1: Tools applied by previous researchers

No	Tool	Authors	Field of Research	Feature Uses	Weaknesses
1	Apktool	[18]	Malware Detection	-Permissions -API calls	Manual
2	Apktool	[27]	Malware Detection	-Permissions	Manual
3	Apktool	[28]	Categorization and detection	-Permissions -API level	Manual
4	Apktool	[29]	Security Threshold	-Permissions -String	Manual
5	Apktool	[30]	Malware Detection	-Permissions -API calls	Manual
6	AAPT	[31]	Malware Detection	-Permissions	One apk at a time

Table 1 shows the tools used in order to parse and extract the permissions. In comparison, a Windows batch file or AAPT tools are utilized to extract all request permissions from all APK files at one time. This process would extract all the required permissions coded inside the AndroidManifest.xml and return matched features into one text file. Furthermore, a study [31] has built an automatic extraction of permissions through an application using Python.

Many web based extraction systems are able to extract more than one features at a time. For example, AVC UnDroid can extract several source code features and analyze the apk files. However, this system only allows users to upload not more than 7 Mb size of file. APK Analyzer is based on Joe Sandbox Mobile which performs deep malware analysis for malware targeting mobile platforms. This web based program only allow users to transfer one file at a time and is limited to 20 Mb file size. Other than that, VirusTotal is a well-known android malware analysis program that allows users to upload files up to 128 Mb. Even though these web-based systems are able to do extraction for apk file, they are restricted to one file at a time. For comparison, MAPE is capable to process more than one file and with unlimited apk file size at a time.

Thus far, no mining technique for extracting request permissions and API calls in the Android environment is found. A sequential search in Depth First Search methods was utilized in order to automate the extraction for multiple apk files. Details on this searching technique are elaborated in Section 2.3.

The remainder of this paper is structured as follows: Section 2.0 presents the methodology of MAPE; Section 3.0 illustrates the performance evaluation; and finally, the conclusion in Section 4.0.

2.0 METHODOLOGY

Currently, researchers use several tools such as ApkTool, AAPT, dex2jar, Dex and JD-GUI separately to catch the wanted API calls as shown in Figure 1. Basically, an APK file will be decompressed using APKTool. Only then, all the permissions be extracted using the AAPT. In order to convert and view the java source codes of an application, dex2jar and JD-GI are used. Then, one can mine the source codes to match the targeted features before obtaining results. The manual process of extracting and mining API calls is shown in Figure 1.

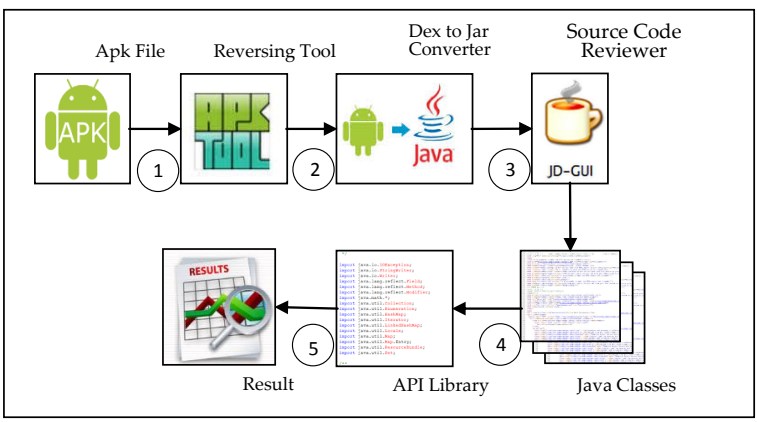

Figure 1: Manual process of extracting and mining API calls

The method was enhanced by integrating several operations (processes 1 to 4) to automatically catch the permission and desired API calls. Hundreds of apk files can be processed together using a single program, producing the necessary outputs. Figure 2 illustrates the MAPE process.

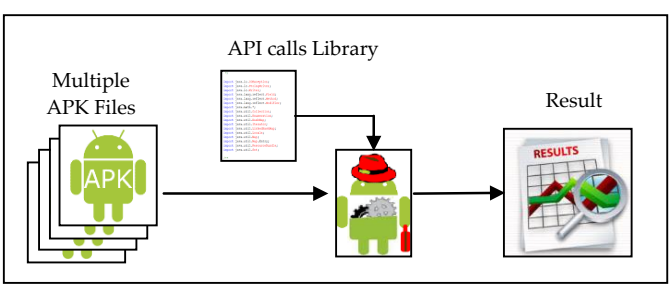

Figure 2: Multiple Android Package Extractor Process

2.1 MAPE Flowchart

Currently, MAPE can fully run using Windows operating system because the batch file created would utilize the shell program and read the file, then execute commands line-by-line. This .bat extension file is used in DOS and Windows. However, MAPE can run smoothly on other platforms if the extractions of permissions are excluded. Figure 3 illustrates the flowchart of MAPE.

Firstly, MAPE will detect the operating system used by a user. MAPE will dump all the request permissions located in AndroidManifest. xml if Windows operating system is detected. This code can simply be coded using a text file program such as Notepad or Notepad++. In order to execute this code, the AAPT.exe must be placed into the same folder where the apk files are located. MAPE will continue running the program even though the operating system is not Windows but the request permissions in APK will not be extracted.

In order to extract the API calls, the system will rename the .apk file to .zip first before unzipping it to produce the classes.dex file. Then, this tool will delete classes.dex file right after generating the .jar file using dex2jar tool. Extracting .jar files into a folder will be the next process executed by MAPE in order to get the .java files. MAPE will read through all the .java files in the extracted folder and compare it with the entries in API calls text file (library). Afterward, the system will generate (for first time) or amend result.txt file which has the filtered list of API matched with the library's. Lastly, the system will delete all the temporary files and proceed with the second APK until the last APK file is processed.

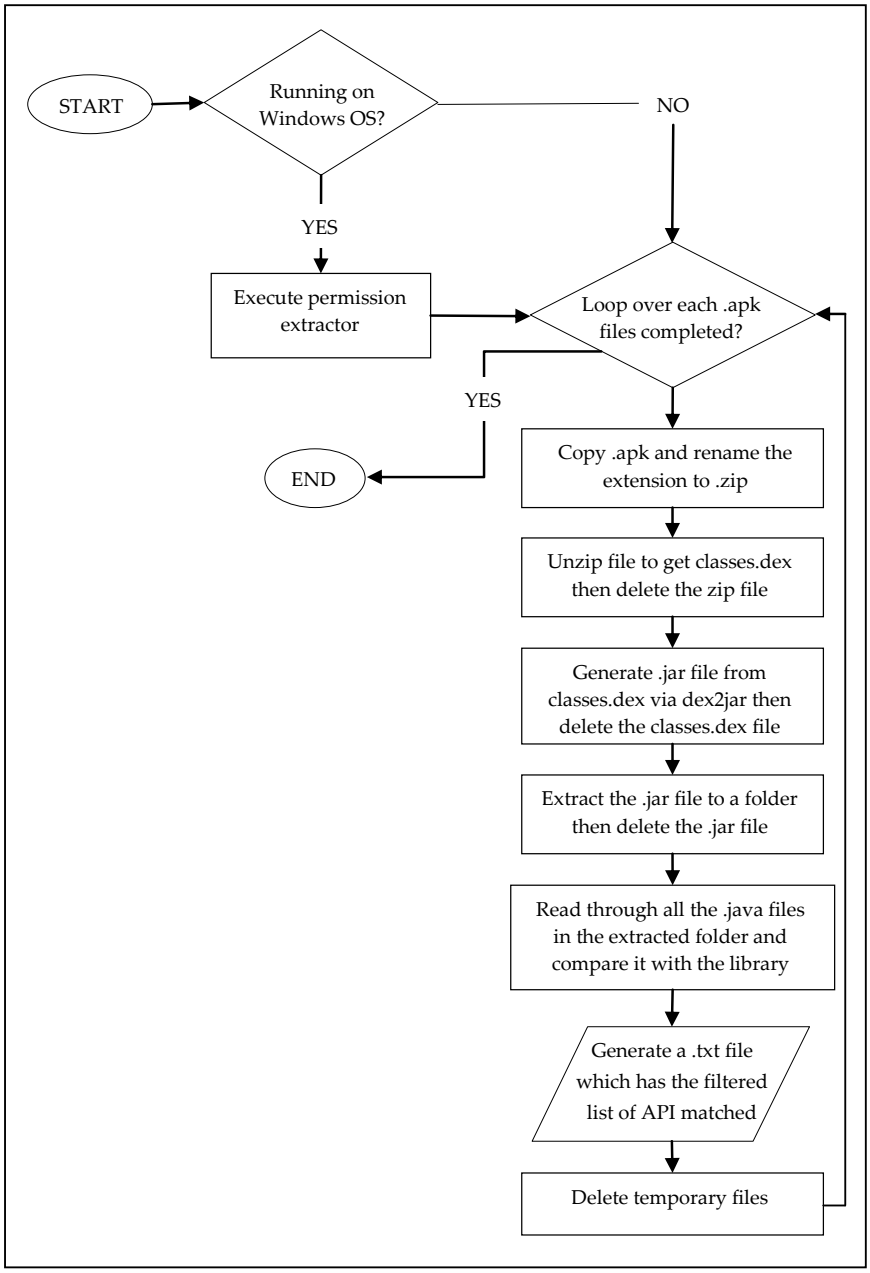

Figure 3: Flowchart of MAPE

2.2 Extracting API Calls

Depth First Search (DFS) searches deeper into the problem space and uses last-in first-out stack for keeping the unexpected nodes. More commonly, DFS is implemented recursively with the recursion stack taking the place of an explicit node stack. Even though there is a possibility that it may go down the far left path forever and is not guaranteed to find the solution, this technique is easy to be implemented and using less memory. Moreover, DFS is one of the most versatile sequential techniques [32]. This approach was utilized as this technique would traverse all the classes through all java files to catch the targeted API calls as shown in Figure 4. Hence, no targeted classes are exempted.

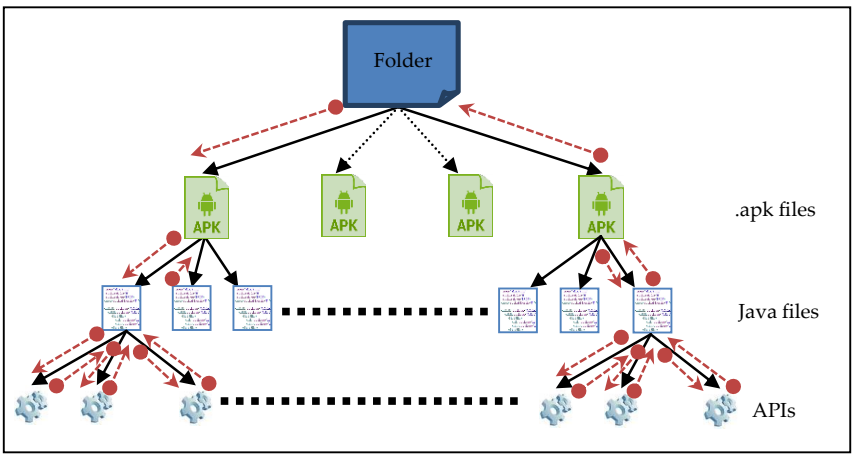

Figure 4: DFS Traversing Tree for Extracting Android API

2.3 Sequential Searching

Basically, API calls is a combination of two or more subroutines or protocols that is delimited by full stop (dot). In this study, the whole combination is integrated and becomes a keyword for search purposes. For example, android.net.wifi. WifiManager is a combination of four protocols which provides the primary API for managing all aspects of Wi-Fi connectivity. This API call was captured as a keyword and then was compared with the desired API lists in the library. Linear search or sequential search (citation) technique based on keywords was applied in order to complete the tasks. Linear search would sequentially checks each element of the list until it finds an element that matches the target value. The search terminates unsuccessfully if the algorithm reaches the end of the list [33]. Figure 5 is the algorithm of DFS and sequential search in which to mine the API calls.

```
1.  Get A
2.  For each in A
    2.1    Get C
    2.2    For each in C
         2.2.1   Set i to 0
         2.2.2   If L_i = T ; go to step 4
                           Increase i by 1
                 2.2.3   If i < n; the search terminates successfully; return i.
                         Else, the search terminates unsuccessfully.
```

Figure 5: DFS with sequential search algorithm for MAPE

Given list *L* of n elements with values L_0 … L*n*-1, and target value *T*, the following subroutines used sequential search to find the index of the target *T* in *L*. The outer loop is where *A* = number of apk files, and *C* = number of class files coded for inner loop.

3.0 PERFORMANCE EVALUATION

This study was conducted under the environment of Windows 7 Professional operating system plus Intel Core i7-6700 3.40 GHz CPU and 16GB RAM capabilities. The results for all the tests are shown in Table 2 .

Table 2: Time taken by MAPE

# APKs	# API Calls	Time (minutes)
100	50	44
	100	46
	200	50
500	50	77
	100	85
	200	89
1000	50	113
	100	121
	200	129

A total of three groups of APK files were tested (100, 500 and 1000) using MAPE with different numbers of API calls. Varying file sizes were involved in which the smallest file size was 202KB and the largest was 39.15MB. No error occurred during the testing period. The results showed that there was a slight time increase in processing time when the numbers of API calls were increased. Moreover, MAPE only took less than half a minute per APK when processing 100 files. The results

also showed that the completion time for each file is decreasing if the number of files is added. Finally, MAPE produced all the request permissions and desired API calls recorded in a single text file.

To extract and record request permissions manually, one needs to spend about 2 to 4 minutes for individual APK using AAPT through command prompt. Mining related API calls is another step and consumes more time. With MAPE, the processes time can be shorten. MAPE is also able to execute both processes without restriction to number of files and file size. Even though web based extractor is available, it cannot process more than 1 file at a time and is limited to APK file size.

4.0 CONCLUSION

Android Package Kit (APK) is a package file specifically in zipped format based on JAR file format. This package contains several folders and hundreds of files for the installation of mobile apps running on Android operating system. Mining codes from a large number of files may take a long time without the proper tools and algorithm. MAPE is an automation tool for searching the codes inside hundreds of android apk files. MAPE may shorten the time for processing and mining permissions and API calls in android apk files. Thus far, there are no similar tools to MAPE. Existing tools such as ApkTool, dex2jar and JD-GUI concern with the development of MAPE. Additionally, DFS is applied with sequential search on keyword in order to capture the related search items. As a start, MAPE can only look into permissions and API calls. In the future it can be extended to mining several other features in Android apk files. By introducing MAPE, working hours on research related to static code analysis on Android environment can be shortened.

ACKNOWLEDGEMENT

The authors would like to thank Universiti Teknologi Mara, Universiti Teknikal Malaysia Melaka, Kementerian Pendidikan Malaysia (Higher Education) for their facility usage and financial support.

REFERENCES

[1] A. Sharma and S. K. Dash, "Mining API Calls and Permissions for Android Malware Detection," in International Conference on Cryptology and Network Security, Crete, Greece, 2014, pp. 191–205.

[2] D. Uppal, V. Mehra and V. Verma, "Basic survey on Malware Analysis, Tools and Techniques", *International Journal on Computational Science & Applications*, vol. 4, no. 1, pp. 103–112, 2014.

[3] R. Raveendranath, V. Rajamani, A. J. Babu and S. K. Datta, "Android malware attacks and countermeasures: Current and future directions," in International Conference on Control, Instrumentation, Commumincation and Computational Technology, Kanyakumari, India, 2014, pp. 137–143.

[4] W. Zhou, Y. Zhou, X. Jiang and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in Second ACM conference on Data and Application Security and Privacy, Texas, USA, 2012 pp. 317-326, 2012.

[5] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp and J. Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques", *International Journal of Information Security*, vol. 14, no. 2, pp. 141–153, 2015.

[6] A. Feizollah, N. B. Anuar, R. Salleh and A. W. A. Wahab, "A review on feature selection in mobile malware detection", *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[7] Z. Fang, W. Han and Y. Li, "Permission based Android security: Issues and countermeasures", *Computer Security*, vol. 43, pp. 205–218, 2014.

[8] S. Feldman, D. Stadther and B. Wang, "Manilyzer: Automated Android malware detection through manifest analysis," in International Conference on Mobile Ad Hoc and Sensor Systems, Philadelphia, USA, 2015, pp. 767–772.

[9] T. Kanda, Y. Manabe, T. Ishio, M. Matsushita and K. Inoue, "Semi-automatically extracting features from source code of android applications", *Transactions on Information and Systems*, vol. E96–D, no. 12, pp. 2857–2859, 2013.

[10] M. Frank, B. Dong, A. P. Felt and D. Song, "Mining Permission Request Patterns for Malicious Android Applications," in International Conference on Data Mining, Brussels, Belgium, 2012, pp. 870–875.

[11] K. Wain, Y. Au, Y. F. Zhou, Z. Huang and D. Lie, "PScout : Analyzing the Android Permission Specification," in 2012 ACM conference on Computer and communications security, North Carolina, USA, 2012, pp. 217–228.

[12] M. Linares-Vasquez, C. McMillan, D. Poshyvanyk and M. Grechanik, "On using machine learning to automatically classify software applications into domain categories", *Empirical Software Engineering*, vol. 19, no. 3, pp. 582–618, 2014.

[13] B. Olabenjo. (2016). *Applying Naive Bayes Classification to Google Play Apps Categorization* [Online]. Available: https://arxiv.org/pdf/1608.08574.pdf

[14] S. M. A. Ghani, M. F. Abdollah, R. Yusof and M. Zaki, "Recognizing API Features for Malware Detection Using Static Analysis", *Journal of Wireless Networking and Communications,* vol. 5, no. 2A, pp. 6–12, 2015.

[15] P. Pearce, A. P. Felt, G. Nunez and D. Wagner, "AdDroid: Privilege separation for applications and advertisers in android," in 7th ACM Symposium on Information, Computer and Communications Security, Seoul, Korea, 2012, pp. 71–72.

[16] Y. Zhongyang, Z. Xin, B. Mao and L. Xie, "DroidAlarm: An all-sided static analysis tool for Android privilege-escalation malware," in 8th Symposium on Information, Computer and Communications Security, Hangzhou, China, 2013, pp. 353–358.

[17] P. P. K. Chan and W. K. Song, "Static detection of Android malware by using permissions and API calls," in International Conference on Machine Learning and Cybernetics, Lanzhou, 2014, pp. 82–87.

[18] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, USA, 2013, pp. 300–305.

[19] Q. Qian, J. Cai and R. Zhang, "Android Malicious Behavior Detection Based on Sensitive API Monitoring," in Advanced Science and Technology Letter, Jeju Island, Korea, 2013, pp. 54–57.

[20] H. Zeng, Y. A. N. Ren, Q. Wang, N. He and X. Ding, "Detecting Malware and Evaluating Risk of App Using Android Permission-Api System," in 11th International Computer Conference on Wavelet Active Media Technology and Information Processing, Chengdu, China, 2014, pp. 440–443.

[21] X. Wang, J. Wang and Z. Xiaolan, "A Static Android Malwar Detection Based on Actual Used Permissions Combination and API Calls", *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 10, no. 9, pp. 1486–1493, 2016.

[22] H. Zhong, T. Xie, L. Zhang, J. Pei and H. Mei, "MAPO: Mining and Recommending API Usage Patterns," in Genoa Proceedings of the 23rd European Conference, Genoa, Italy, 2009, pp. 318–343.

[23] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee and K. P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in 7th Asia Joint Conference on Information Security, Tokyo, Japan, 2012, pp. 62–69.

[24] K. Iwamoto and K. Wasaki, "Malware classification based on extracted API sequences using static analysis," in Asian Internet Engineeering Conference, Bangkok, Thailand, 2012 , pp. 31–38.

[25] M. Zhao, T. Zhang, J. Wang and Z. Yuan, "A smartphone malware detection framework based on artificial immunology", *Journal of Networks*, vol. 8, no. 2, pp. 469–476, 2013.

[26] Y. Xiaohui, S. Yubo and C. Fei, "Android S Sensitive Data Leakage Detection Based on Api Monitoring," in Fifth International Conference on Multimedia Information Networking and Security, Beijing, China, 2013, pp. 907–910.

[27] U. Pehlivan, N. Baltaci, C. Acarturk and N. Baykal, "The Analysis of Feature Selection Methods and Classification Algorithms in Permission Based Android Malware Detection," in IEEE Symposium on Computational Intelligence in Cyber Security, Florida, USA, 2014, pp. 1–8.

[28] V. Grampurohit and V. Kumar, "Category Based Malware Detection for Android," in Security in Computing and Communications, Delhi, India , 2014, pp. 239–249.

[29] J. Jeon, K.K Micinski, J.A Vaughan, A. Fogel, N. Reddy, J.S Foster, T. Millstein "Dr. Android and Mr. Hide: fine-grained permissions in android applications," in Second ACM workshop on Security and Privacy in Smartphones and Mobile Devices, Raleigh, USA, 2012, pp. 3–14.

[30] D. Upadhyay, M. Munghate, S. Dharbey, and A. Bondre, "Detecting Malicious Behavior of Android Applications", *International Journal of Science Technology & Engineering*, vol. 2, no. 10, pp. 663–668, 2016.

[31] P. Rovelli and Y. Vigfusson, "PMDS : Permission-based Malware Detection System," in 10th International Conference on Information Systems Security, Hyderabad, India, 2014, pp. 338–357.

[32] J. H. Reif, "DFS is Inherently Sequential", *Information Processing Letter*, vol. 20, no. 5, pp. 229–234, 1985.

[33] D. E. Knuth, "*Sorting and Searching: The Art of Computer Programming.* Reading, MA: Addison-Wesley Professional, 1998.