International Journal of Computer Networks and Applications (IJCNA)
DOI: 10.22247/ijcna/2019/187292 Volume 6, Issue 5, September – October (2019)

REVIEW ARTICLE

A Review of Static Malware Detection for Android Apps Permission Based on Deep Learning

Hamida Lubuva
School of Information and Communication Engineering, University of Science and Technology Beijing, Beijing, China.
hamidalubuva@gmail.com

Qiming Huang
School of Information and Communication Engineering, University of Science and Technology Beijing, Beijing, China.
qmhuangcn@163.com

Godfrey Charles Msonde
School of Economics, Renmin University, Beijing, China,
gmsonde@gmail.com

Published online: 31 October 2019

Abstract – In recent years, Android has been the main mobile operating system. The proliferation of apps powered not only by Android magnetized app developers, but also by malware developers with criminal intent to design and distribute malicious apps that can influence the ordinary activity of Android phones and tablets, steal private information and credentials, or even worse, lock the phone and ask for ransom. This study was carried out with a view of bring out clearly the review of previous researches carried regarding static analysis and pinpoint out what to be done in future. A systematic literature review which involves studying 56 research papers published in regard to static analysis. This review elaborate permissions misuse, reverse engineering and concept of static analysis in general. The outcomes of the review revealed that static analysis is widely used since it is not performed at run-time hence malicious applications cannot access to the device during analysis unlike dynamic analysis. During the review no single work done to the satisfaction curbing the existing and future evolving malwares. This study will help academicians to gain insight concerning static analysis without extensively perusing several articles to understand static malware analysis based on deep learning.

Index Terms – Static Analysis, Reverse Engineering, Permissions, Manifest File, APK File, Malicious Applications.

1. INTRODUCTION

With the rapid emergence of android as the principal operating system has led to the rise of malware applications built by hackers with a view of extracting both personal and sensitive user's data on the android device for malicious purposes [1], [2]. Static analysis has been a prime method used since the extraction of manifest file information is done before the installation of the application unlike both dynamic and hybrid analysis [3].

The main aim of this study is to review previous work done on static malware of android apps. The subsequent sections depict the research done, citing the advantages and shortcomings. According to [4], discussed permission misuse by android apps using a static analysis tool of identification stating that it is possible to obtain all the manifest file permission. Despite the study, the paper did not mentioned or show the dataset used during the analysis and the procedure of getting readable contents of a minfest.xml file. Similarly, [5] studied system permission to show whether the application is over privilege but loopholes exists in the write-up since the author mentioned repacking of application files without executing on the device but there is no dataset and analysis performed based on static feature extraction.

According to [6], the authors proposed heuristic model and compared adagio, Drebin, ISCX and Mamadroid on a dataset and found their proposed model outpeforms the others. The only ad-hoc facing their framework is the proper selection of malware models to preserve great detection rate and apposite runtime performance as the method discovery analysis considerably depends on the malevolent applications used for model excavation. Likewise, [4], based the research on the SharedUserID by comparing the certificates of two applications if both have same features then the apps are likely to share same permissions using an android security tool. This was done in order to deduce whether one application misuse the permission of another application if granted access by the user. Althought the author did not provide the data used to test how the algorithm of proof of concept was implemented using a security tool as dipicted in the work done by Karthick and Binu [7].

Conversely, [8] deliberated on malware for smart phones in overall. Though, the paper deliberates various categories of features very concisely and the authors did not cover all kinds of obtainable features. Accordingly, [9] explores numerous types of smart devices available for malicious applications, their effect on apps and related discovery approaches with a 99.8 percent accuracy detection rate. Nonetheless, they did not indicate what features they used in detection, bearing in mind that features have substantial influence on detection. In [10] the authors review diverse analysis procedures in smart devices illegal programs detection. The paper outline the examples of detection approaches along with their explanation.

The paper doesn't embrace what datasets used and valuation measures. Additionally, it does not illustrate all the latest works extensively. According to Peng et al,.[11], they scrutinize advancement of mobile malware, damages they cost and their proliferation model. Different operating systems are accounted for in the paper making it difficult to carefully review all available types. Conversely, we emphasize static malware detection for Android Apps Permit to mitigate how best the method is compared to dynamic, hybrid or metadata malware detection and prevention methods where the application is run on the user device without the user's knowledge of what is actually happening on the background.

The remainder of this paper is organized as follows. Section 2 offers background information on android and static analysis needed for paper repository by discussing definitions of static program analysis, model permission and techniques of analysis. Section 3 presents a summary of related work using static analysis presentations based on permissions from android apps. Section 4 addresses the description of android apps in reverse engineering. Section 5 outlines the debates and the paper is concluded.

2. BACKGROUND INFORMATION ON ANDROID AND STATIC ANALYSIS

In order to gain understanding of the purpose of this study, we review and give the reader the required preliminary information on android and static analysis. We explain the concept of static program analysis, permissions and analysis technique.

2.1. Concepts of Static Program Analysis

Static code analysis is also called static program analysis, which means that the application under test cannot be conducted dynamically and that it can detect bugs in an early stage before it is implemented [12]. The opposite of static code analysis is dynamic code analysis. In the latter, the program is executed and developers look for run-time errors as stated by [13].

According to Ghahrai [14], after coding and before performing unit tests, static analysis is performed. Static testing can be performed by a machine to "pass" the source code automatically and to detect non-compliance rules. A compiler that finds lexical, syntactic and even semantinal errors is the classic example.

Static software analysis usually includes an automated method that uses inputs the source code or object code of a program, analyses this code without it being executed, and produces results by analyzing its code structure, sequences of statements, and how variable values are interpreted via the divergent function calls. Static analysis ' primary benefit is that it can disclose mistakes (or vulnerabilities) that do not appear (or are not exploited) until long after the software is published to the public. There are different benefits/advantages of static analysis as follows; helps recognize prospective software quality problems before the software enters manufacturing during the development stage [15]. It identifies code regions that need to be re-factored / simplified [15].By concluding it for software to work and developers to comprehend their software, static code analysis is not only helpful but also essential. It simplifies the search process for bugs and mistakes by pointing to them correctly and helps to define problems.

2.2. Permissions

The resolution of a permission is to safeguard the privacy of an android user [16]. Not all permissions are dangerous some are useful to the developer to design security of an android device. Mobile apps must appeal for permission to access user's sensitive data for instance short messages and phone contacts, as well as particular system features for instance internet and camera. Reliant on the feature, the system might allow the permission spontaneously or might occassion the user to accept the request. A fundamental design of the android security design is that no mobile app, by default, has permission to accomplish any operations that would unfavorably impact other applications or the user. Android apps must adopt the least privilege to minimize damages [17]. Table 1 below depicts other previous studies in regard to android permission which are normally misused.

Android. Permission	Usage	Exploitation
<READ/WRITE_ EXTERNAL_ STORAGE>	Permit to read or write device's external storage	Malicious app can read sensitive data of the user and write its malicious code on the device external storage.

REVIEW ARTICLE

<CALL_PHONE>	Permits an app to induce a phone call without going via the interface of the user dialer for the user to authorize the call being engaged.	Aid the user to record the voice of the user and use it for malicious purpose
<RECEIVE_SMS>	Permits an app to observe the inbound SMS messages, to record or implement processing on them.	Aid the malware to read, write and receive user's sensitive information to the malicious app developer.
<SET_PROCESS_LIMIT>	Permits an app to fix the determined number of app's processes that can be running but not required.	Overwhelming the device memory thus rendering to slowness in its normal operation
<ACCESS_ WIFI_ STATE>	Permits the app to access data about Wi-Fi network connected	Can aid the malware in hacking the Wi-Fi network and transmitting user information by utilizing this info.
<READ_ PHONE_ STATE>	Offers access to personal information of phone like IMSI/IMEI device identifier, Voice Mail Box, Phone Number, SIM ID etc.	Aids the developer of the malicious app to keep track of user's device and can include user's device in malevolent activities using information gathered.

Table 1 List of Dangerous Android Permission [18]

2.3. Analysis Technique

Control-flow analysis: Determining the order of execution of program statements or instructions. The control sequences are usually displayed as a control-flow graph (CFG). The CFG specifies all feasible routes of execution [19].

Important control flow constructs:

Method calls: program analysis to define the function calls receiver – e.g., virtual functions, function pointers: abstract interpretation, type structures and restriction.

Basic block: Maximum sequence of successive statements with one entry point, one exit point and no inner branches.

Loops: An iteration block of codes till a specified state is achieved.

Data-flow analysis: Is a monitoring technique for how variables and values change through the flow of the program. It is a method for collecting data on the feasible set of values calculated at different points in a computer program. The control flow chart (CFG) of a program is used to determine those components of a program that could be propagated by a specific value allocated to a variable [20]. Compilers often use the data collected when optimizing a program for instance:

$x = c + d;$

$x = 10 * 7;$

It is easy for an optimizer to recognize that: a "useless" assignment is the first assignment tox, since the calculated value for x is never used (and thus the first statement can be removed from the program). At compile time, the expression $10 * 7$ can be calculated, simplifying the second assignment statement to $x = 70$;

Points-to analysis: involves computing a static abstraction of all the data to which a pointer expression (or just a variable) may point during runtime of the program [21].

3. SUMMARY OF RELATED WORK BASED ON PERMISSIONS OF ANDROID APPS USING STATIC ANALYSIS

Nearly 80 percent of the papers were based on static analysis as illustrated by the study. There is a lot of job done in static

REVIEW ARTICLE

analysis as described Table 2, and more needs to be explored to get greater precision with a minimum amount of characteristics, as fewer characteristics decrease regression and classification training and testing time and provide quicker reaction [22].

There are different reviews which talked about static malware detection and their challenges. A framework for automatically analyzing permission use in Android apps was suggested and created. Permlyzer can analyze the use of permissions in Android apps accurately and thoroughly [23].Defines a strategy that uses system call log data to create a dataset [24]. This paper tackles the issue of android malware intrusion. Use Rotation Forest in this article to tackle the issue of android malware intrusion [25] .Table 2 provides a summary of the previous work based on static analysis.

Ref	Mechanism	Malware Detection Rate/ Accuracy	Strengths	Weakness
[26]	Stowaway	Not Stated	The authors tested 950 applications and used stowaway tool to detect over-privilege android apps.	Despite using Stowaway , the tool is incapable of handling some multifaceted reflective calls.
[27]	Stowaway/ StackOverflow	Not stated	This paper illustrated that the authors tested 10,000 apps and offered statistical models for envisaging permission abuse and call for permission documentation. They instituted that the popularity of a permission is sturdily related with its abuse, while other aspects such as effect and intrusion had little effect.	In this paper, Stowaway has been used together with Stack-Over-flow. But according to [26], is incapable of handling some complex reflective calls
[28]	Web-based app	Not stated	The paper analyses 500000 apps for mobile device and the privileges they request using an algorithm of machine learning in order to rate the risk of an app for developers and user to adopt.	The authors only mentioned that the apps were extracted and permission analyzed but no tool mentioned or software that helps the developers or users to extract the APK file of an android app.
[29]	FAMOUS	99%	This paper proposes a predictive	The authors tested applications and

REVIEW ARTICLE

			forensic approach to detect suspicious android applications based on a trained model called Forensic Analysis of mobile devices using scoring (FAMOUS) which is intelligent to scan all the apps installed in the attached device and offer a descriptive report. 11,371 apps tested.	showed some suspicious APK file but they did not mentioned the dataset. They did not mentioned how many apps were tested to the devices attached to FAMOUS. The procedure of testing apps using FAMOUS was not shown hence difficult for the developers or users to attest the tool.
[30]	Permission Management App	Not stated	In this scheme, end-users can avert malware behavior from retrieving sensitive data and invoking sensitive API in real time. The explanation	The method incurs extra performance cost. Their proposed scheme was not tested on a real mobile device hence their results

			upsurges the flexibility of managing permission and advances the security and consistency of data in mobile devices. The authors analyzed 2,200 apps.	can be biased in the real world.
[31]	APK Auditor	88%	The authors tested 8,762 apps and classified them as malicious or benign successfully.	The method employs the APK client, Database signatures and the central server. The disadvantage to this method is that extra cost is incurred in setting up the tool for analysis.
[32]	RefinedDroid	Not stated	In this paper the authors tested 727 apps to attest for fine-grained permission model which are	The tool deployed in this research is that it modifies the APK file of application during feature

REVIEW ARTICLE

			appropriate for many standard apps. This tool allows the user to lower the privilege level of permission.	extraction. This may lead to exaggerated outcomes.
[33]	APK Analyzer	Not stated	The authors supplied the dataset of 576,174 android apps while conducting the experiment for free android app and found there method to be better than those used by [34] which contains possible flaws that cause imperfections.	The authors did not test paid applications to determine effectiveness of their methodology.
[35]	Java-based custom built APK analyzer	90%	In this paper the authors develop and examine proactive Machine Learning approaches based on Bayesian classification intended to uncover unidentified Android malware. The authors tested 2000 permission-based framework.	The authors used a small sample size of malware apps in there experiment which might not be compared to authors using large samples in the experiment to depict model performance.
[36]	DroidRay	Not stated	The authors tested 24,259 apps to show malware based on geographical spread. The model helps to curb new form in which malware spreads out.	Their model is not good enough since it did not mention the malware detection rate as compared to previous studies.
[37]	Weka tool	Not Stated	The authors used the machine based learning tool to test 200 apps to extract permissions in order to examine	The authors tested a small sample size and they authors did not classify the malwares whether

REVIEW ARTICLE

			whether they are malwares.	they are infosteal, Trojan among others.
[38]	SherlockDroid, (Alligator)	98.04%	The authors tested 102,156 apps using Alligator which is able to uncover unknown malware.	The authors failed to explore new clusters and learning scripts. The authors abscond introducing weights on properties so that algorithms such as deviation do not deliberate each property with alike status
[39]	Apposcopy	Not stated	The authors evaluated their tool on a mass of available Android applications and attest that it can efficiently and reliably to determine malware that fit to definite families.	In this scheme it is difficult to design any signature oriented scheme like Apposcopy since it can be defeated by obfuscation scheme such as real coding.

			The authors tested 11,215 apps and found 16 of them to be malwares.	
[40]	Woodpecker	Not Stated	The author used their tool employing inter-procedural data flow analysis scheme to exhaustively uncover possible capability disclosures where an unreliable app can acquire unlawful access to subtle data or restricted actions.	The authors did not provide the dataset to assist in future studies. They only stated they tested 13 permissions in which 11 were leaked
[41]	DroidAPIMiner	99%	In this paper, the authors purpose to alleviate malware installation via providing vigorous and frivolous classifiers. The authors	The authors in their experiment realized a big false positives and negatives of 2.2% but they did not attest where the problem

REVIEW ARTICLE

			extensively carry out an analysis to excerpt pertinent features regarding malware behavior netted at API level and appraised different classifiers by the produced feature set during the testing of 3987 apps.	occurred during their analysis.
[42]	DNADroid	Not stated	The tool adopted by authors in this study showed a very low false positive rate and confirmed that all 141 apps identified by DNADroid are indeed replicas through visual or behavioral confirmation.	The authors did not mentioned the malware detection rate in their study.
[43]	DroidChameleon	Not stated	The authors found out that all the anti-malware products are susceptible to common alterations.	The authors failed to strength their studies but depicting the detection rate to show best their model is as compared to other existing literature.
[44]	CHEX	Not stated	The scholars presented a method to spontaneously discern entry points in mobile app, as well as the new analysis model and attest that app splitting is effective and perfect way to model executions of manifold entry points and expedite universal data-flow scrutiny. The authors tested 5486 real-world	This study did not put into consideration the rate at which the malicious apps were discovered to support their study.

REVIEW ARTICLE

			apps.	
[45]	AndroSimilar	94.4%	The authors were able to discover malicious apps vigorously using signature statistics in feature selection by deploying relationship digest hashing scheme.	In their approach the authors did not consider memory constrain in developing a robust family signature to identify variants with family illustrative e signatures
[46]	DroidBarrier		The authors achieved high reassurance by designing an verification model that uses protected application IDs, preserved and secured by system runtime, to validate processes and put together their identity to genuine apps installed on the	In their work the authors not focus on validating inter-process communications and authorizing admission to apps' assets.

			android.	

Table 2 Strength and Weaknesses of selected related work based on permissions of android apps using static analysis

4. REVERSE ENGINEERING OVERVIEW OF ANDROID APPLICATIONS

According to [47], Reverse engineering is the analysis of a subject system for classifying system components and their interrelationships, and for providing a specific or higher degree of perception image of a process. Reverse engineering is just the process of examining the code-to-code but not changing or replicating the source codes [22].To perform inverse engineering .apk file needs to be decompiled which offers Dex and Android manifest file in an indecipherable format [48].

A number of arsenals are accessible for reverse engineering such as JD-GUI, DEX2JAR [49], APKTOOL [50], AXMLPRINTER2.JAR [51], ANDROGAURD[52] and CLASSYSHARK [53]. Figure 1 depicts APK file conversion steps to obtain the original source code of the manifest file, resources and the java codes.

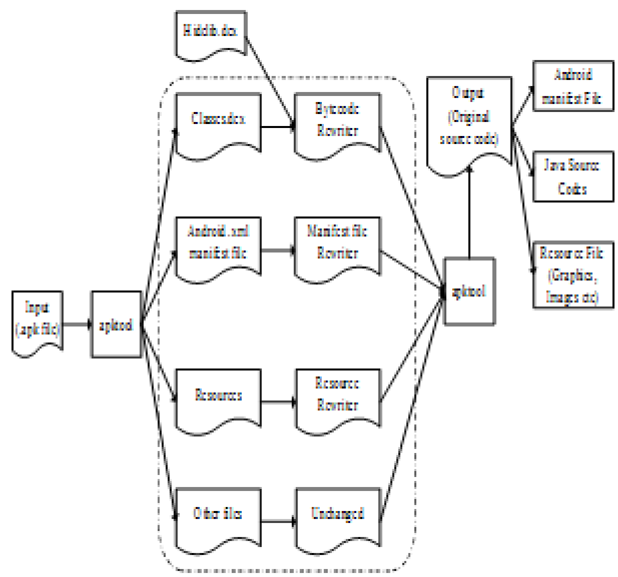


Figure 1 Reverse Engineering of APK File

In reverse engineering the APK file is extension is change from .apk to .zip and its contents extracted to obtain meta-inf, res, classes.dex, resources.arsc and the AndroidManifest.xml [54]. The contents are then extracted to a specified folder for analysis in the subsequent processes. The apktool.jar and the apktool.bat are utilized in a command prompt to obtain the manifest file and the java codes used as well as other resources as shown in Figure 1 which has been modified from the Pooja Singh et al. reference [55]. This process will enable

REVIEW ARTICLE

the user to study and analyze the permission model required by the application in order to install it unlike the dynamic analysis where the android application accessed the user's device for the user to know the permission [56].

5. DISCUSSION

In this paper, a review of about 48 articles were studied regarding to static analysis in Android malware detection based on manifest file which contain permissions a user must accept before installing the application. Most of the android applications are created by hackers in order to steal private information of a user on the device. Previous researched studied in this work reveals the android manifest.xml file contain information that an application intend to do. For instance several manifest files contain the READ_CONTACT which reads and store all the personal contact of a user. These contacts will be used by the hackers for their ill intentions. In the internet several applications have been developed with productive names to perform a certain tasks but the android app does some different tasks like stealing photos and images from the user's phone. In this view it is important to analyze the APK android file before allowing it to access to the user device. Static analysis is the best feature as compared to dynamic, hybrid and metadata where the APK conversion is done before installing the file to the user's device. APK file cannot be studied without decompiling it to get the original source code. Since the manifest file contain the permission like WRITE_EXTERNAL_STORAGE, this permission will allow malware such as adware and other dangerous ones to be installed onto the user device and this will use the phone read access memory thus making the phone slow. Some of these adware are able to send and several SMS if the SEND_SMS and RECEIVE_SMS permissions are enabled. In this study, static analysis is recommended as its plusses outperforms the other APK analysis features.

6. CONCLUSION

In this work, the use of static malware detection in android malware apps was thoroughly reviewed. A comparison of current job has been provided with regard to certain criteria. The review identified knowledge gaps in the current job, highlighting significant problems and opening problems that will guide future study initiatives. Analysis of static Android malware dominates the current job. Future work may consider reviewing other methods such as dynamic studies or the use of methods of hybrid assessment or deployment of metadata. Except in a few cases, sharing research datasets and tools among researchers lingered unaddressed. Hardening deep learning models against various assaults on adversaries and detecting, describing and measuring concept drift are essential in future job on malware detection for Android. In addition, scientists need to bear in mind the restriction of deep learning techniques such as absence of transparency and its non-autonomous model for building more effective models. Finally, the findings of this job can help encourage Android malware detection studies based on techniques of deep learning.

REFERENCES

[1] Z. Fang, W. Han, and Y. Li, "Permission based Android security: Issues and countermeasures," Comput. Secur., vol. 43, no. 0, pp. 205–218, 2014.

[2] F. Tchakounté, "Permission-based malware detection mechanisms on android: analysis and perspectives," J. Comput. Sci., vol. 1, no. 2, pp. 63–77, 2014.

[3] M. Egele, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools Vienna University of Technology," ACM Comput. Surv. 44.2, vol. V, pp. 1–49, 2012.

[4] S. Karthick and S. Binu, "Static analysis tool for identification of permission misuse by android applications," Int. J. Appl. Eng. Res., vol. 12, no. 24, pp. 15169–15178, 2017.

[5] D. Geneiatakis, I. N. Fovino, I. Kounelis, and P. Stirparo, "A Permission verification approach for android mobile applications," Comput. Secur., vol. 49, pp. 192–205, 2015.

[6] A. Skovoroda and D. Gamayunov, "Securing mobile devices: Malware mitigation methods," J. Wirel. Mob. Networks, Ubiquitous Comput. Dependable Appl., vol. 6, no. 2, pp. 78–97, 2015.

[7] S. Karthick and S. Binu, "Android security issues and solutions," IEEE Int. Conf. Innov. Mech. Ind. Appl. ICIMIA 2017 - Proc., no. February, pp. 686–689, 2017.

[8] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," IEEE Commun. Surv. Tutorials, vol. 16, no. 2, pp. 961–987, 2014.

[9] M. La Polla, F. Martinelli, and D. Sgandurra, "A Survey on Security for Mobile Devices," IEEE Commun. Surv. Tutorials, vol. 15, no. 1, pp. 446–471, 2013.

[10] S. Mohite and P. R. Sonar, "A survey on mobile malware: war without end," Int. J. Comput. Sci. Bus. Informatics, vol. 9, no. 1, pp. 23–35, 2014.

[11] S. Peng, S. Yu, and A. Yang, "Smartphone Malware and Its Propagation Modeling: A Survey," IEEE Commun. Surv. Tutorials, vol. 16, no. 2, pp. 925–941, 2014.

[12] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, and R. Maskeliunas, "Android Malware Detection: A Survey," Commun. Comput. Inf. Sci., vol. 942, no. 2, pp. 255–266, 2018.

[13] N. DuPaul, "Static Analysis vs Dynamic Analysis | Veracode," VERACODE, 2019. [Online]. Available: https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing. [Accessed: 25-Oct-2019].

[14] A. Ghahrai, "Static Analysis vs Dynamic Analysis in Software Testing," Testing Excellence, 2018. [Online]. Available: https://www.testingexcellence.com/static-analysis-vs-dynamic-analysis-software-testing/. [Accessed: 25-Oct-2019].

[15] P. Anderson, "The use and limitations of hearing aids," J. Def. Softw. Eng., no. 6, pp. 19–21, 2008.

[16] M. Derks, "Fair Privacy : Improving Usability of the Android Permission System," 2015.

[17] J. Reardon et al., "50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System," 28th USENIX Secur. Symp., pp. 603–620, 2019.

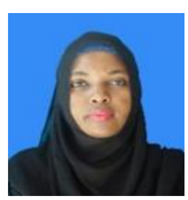
[18] M. Sujithra and G. Padmavathi, "Enhanced Permission Based Malware Detection in Mobile Devices Using Optimized Random Forest Classifier with PSO-GA," Res. J. Appl. Sci. Eng. Technol., vol. 12, no. 7, pp. 732–741, 2016.

[19] F. E. Allen, "Control flow analysis," Proc. a Symp. Compil. Optim., pp. 1–19, 1970.

[20] K. D. Cooper and L. Torczon, "Chapter 9 - Data-Flow Analysis," in Engineering Compiler, K. D. Cooper and L. B. T.-E. a C. (Second E. Torczon, Eds. Boston: Morgan Kaufmann, 2012, pp. 475–538.

[21] L. Li et al., "Static analysis of android apps: A systematic literature review," Inf. Softw. Technol., vol. 88, pp. 67–95, 2017.
[22] S. R. Tiwari and R. U. Shukla, "An Android Malware Detection Technique Using Optimized Permission and API with PCA," Proc. 2nd Int. Conf. Intell. Comput. Control Syst. ICICCS 2018, no. Icirca, pp. 134–139, 2019.
[23] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in Android applications," 2013 IEEE 24th Int. Symp. Softw. Reliab. Eng. ISSRE 2013, pp. 400–410, 2013.
[24] B. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," Proc. ACM Symp. Access Control Model. Technol. SACMAT, Jun. 2012.
[25] H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model," Neurocomputing, vol. 272, pp. 638–646, 2018.
[26] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," Proc. ACM Conf. Comput. Commun. Secur., pp. 627–636, 2011.
[27] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by android apps," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 31–40, 2013.
[28] N. Gruschka, L. Lo Iacono, and J. Tolsdorf, "Classification of android app permissions: Tell me what app you are and i tell you what you are allowed to do," Eur. Conf. Inf. Warf. Secur. ECCWS, vol. 2018-June, no. June, pp. 181–189, 2018.
[29] A. Kumar, K. S. Kuppusamy, and G. Aghila, "FAMOUS: Forensic Analysis of MObile devices Using Scoring of application permissions," Futur. Gener. Comput. Syst., vol. 83, pp. 158–172, 2018.
[30] S. Niu, R. Huang, W. Chen, and Y. Xue, "An Improved Permission Management Scheme of Android Application Based on Machine Learning," Secur. Commun. Networks, vol. 2018, pp. 1–12, 2018.
[31] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," Digit. Investig., vol. 13, pp. 1–14, 2015.
[32] J. Jeon et al., "Dr. android and Mr. hide: Fine-grained permissions in android applications," Proc. ACM Conf. Comput. Commun. Secur., pp. 3–14, 2012.
[33] N. Munaiah et al., "Darwin: A static analysis dataset of malicious and benign android apps," WAMA 2016 - Proc. Int. Work. App Mark. Anal. co-located with FSE 2016, pp. 26–29, 2016.
[34] T. K. Chawla and A. Kajala, "Transfiguring of an Android App Using Reverse Engineering," Int. J. Comput. Sci. Mob. Comput., vol. 3, no. 4, pp. 1204–1208, 2014.
[35] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," IET Inf. Secur., vol. 8, no. 1, pp. 25–36, 2014.
[36] M. Zheng, M. Sun, and J. C. . Lui, DroidRay: A Security Evaluation System for Customized Android Firmwares. 2014.
[37] Z. Aung and W. Zaw, "Permission-Based Android Malware Detection," Int. J. Sci. Technol. Res., vol. 2, no. 3, pp. 228–234, 2013.
[38] L. Apvrille, L. Apvrille, and A. S. Industries, "Pre-filtering Mobile Malware with Heuristic Techniques," GreHack 2013, Grenoble, Fr., no. June 2013, pp. 43–59, 2013.
[39] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis," Proc. ACM SIGSOFT Symp. Found. Softw. Eng., vol. 16-21-Nove, pp. 576–587, 2014.
[40] M. Grace, Y. Zhou, Z. Wang, X. Jiang, and O. Drive, "Systematic Detection of Capability Leaks in Stock Android Smartphones," Ndss, 2012.
[41] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android BT - Security and Privacy in Communication Networks," 2013, pp. 86–103.
[42] J. Crussell, C. Gibler, and H. Chen, "Attack of the Clones: Detecting Cloned Applications on Android Markets BT - Computer Security – ESORICS 2012," 2012, pp. 37–54.
[43] V. Rastogi, Y. Chen, and X. Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks," IEEE Trans. Inf. Forensics Secur., vol. 9, no. 1, pp. 99–108, 2014.
[44] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, 2012, pp. 229–240.
[45] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "AndroSimilar: Robust Statistical Feature Signature for Android Malware Detection," in Proceedings of the 6th International Conference on Security of Information and Networks, 2013, pp. 152–159.
[46] H. M. J. Almohri, D. (Daphne) Yao, and D. Kafura, "DroidBarrier: Know What is Executing on Your Android," in Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, 2014, pp. 257–264.
[47] E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," pp. 13–17, 1990.
[48] S. R. Tiwari and R. U. Shukla, "An Android Malware Detection Technique Based on Optimized Permissions and API," Proc. Int. Conf. Inven. Res. Comput. Appl. ICIRCA 2018, no. January, pp. 258–263, 2018.
[49] H. A. Alatwi, "Android malware detection using category-based machine learning classifiers," 2016.
[50] M.-Y. Su and K.-T. Fung, "Detection of android malware by static analysis on permissions and sensitive functions," in 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), 2016, pp. 873–875.
[51] X. Li, J. Liu, Y. Huo, R. Zhang, and Y. Yao, "An Android malware detection method based on AndroidManifest file," in 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), 2016, pp. 239–243.
[52] C. Liu, Z. Zhang, and S. Wang, "An Android Malware Detection Approach Using Bayesian Inference," in 2016 IEEE International Conference on Computer and Information Technology (CIT), 2016, pp. 476–483.
[53] K. Wang, T. Song, and A. Liang, "Mmda: Metadata Based Malware Detection on Android," in 2016 12th International Conference on Computational Intelligence and Security (CIS), 2016, pp. 598–602.
[54] S. Lachure, U. Pagrut, N. Jichkar, N. Khan, and J. Lachure, "Reverse Engineering APKS for Analysis," pp. 268–272, 2018.
[55] P. Singh, P. Tiwari, and S. Singh, "Analysis of Malicious Behavior of Android Apps," Procedia Comput. Sci., vol. 79, pp. 215–220, 2016.
[56] B. Bonné, S. T. Peddinti, I. Bilogrevic, N. Taft, S. Clara, and B. Bonné, "Exploring decision making with Android ' s runtime permission dialogs using in-context surveys This paper is included in the Proceedings of the permission dialogs using in-context surveys," no. Soups, 2017.

Authors



Hamida Lubuva received her Bachelor Electronics and Communication Engineering in 2016 from the, St. Joseph University, Tanzania. She is currently pursuing Master Degree in Information and Communication Engineering at the University of Science and Technology Beijing. Her research areas include; Terminal Detection, Networking, Network security, Data Privacy.

Huang Qiming is an associate Professor and Master Tutor of Beijing University of Science and Technology, Huazhong University of Science and Technology, Postdoctoral Fellow, Department of Computer Science, Zhejiang University, Associate Research Fellow, School of Computer, Beijing

REVIEW ARTICLE

University of Posts and Telecommunications, Visiting Scholar, University of Hong Kong. At the Beijing University of Science and Technology, the School of Computing, the future of the Internet of Things, cloud computing encryption certification and authorization, block chain smart contracts, data privacy protection and deep learning based computer video classification, network intrusion detection direction of scientific research work, and committed to Applications in smart cities, industrial internet, health care information systems, and car networking. He has published more than 50 academic papers, and more than 20 papers have been included in SCI and EI. There are 1 collection of papers, 1 joint, and 4 textbooks. Teaching data structure and algorithm analysis, mobile Internet, communication network security foundation, modern communication security foundation, computer network, software engineering, computer introduction and programming.

Godfrey Charles Msonde; received his Bachelor Degree of Science in Economics in 2012 from Mzumbe University, Tanzania. In 2019, He graduated Masters of National Economics from Renmin University of China, Beijing China. He is currently pursuing Master of International Public Policy at Wilfrid Laurier University, Canada. His research areas include Digital economy, development studies, Big Data, Smart Agriculture to Africa.