JOURNAL OF COMPUTER SCIENCE AND SOFTWARE APPLICATION

Permission-based Malware Detection Mechanisms : Analysis and Perspectives

Franklin Tchakounté*

University of Ngaoundéré, Faculty of Science, Department of Mathematics and Computer Science, PO Box 454 Ngaoundéré, Cameroon.

***Corresponding author**: f.tchakounte@univndere.cm

Abstract:
Android security has been built upon a permission based mechanism which restricts accesses of third-party Android applications to critical resources on an Android device. The user must accept the set of permissions an application requires, before the installation proceeds. This process is meant to inform the users of the risk of installing and using an application on their device; But most often, even when the permission system is well understood, users are not aware enough of the threat endangered, and trust either the application store or the popularity of the application, and accept the installation without trying to analyse the intentions of the developer. Increasingly, one develops approaches aiming to characterise malware with the permissions individually or associatively taken, with machine learning classifiers. The objective in this paper is to investigate in the literature mechanisms for the characterisation and detection of malwares based on the previous aspects. For that, we illustrate what works from these researches, their limitations and promising considerations for future research.

Keywords:
Android; Detection; Investigate; Malwares; Permissions-based

1. INTRODUCTION

With an estimated market share of 70% to 80%, Android has become the most popular operating system for smartphones and tablets [1, 2]. Expecting a shipment of 1 billion Android devices in 2017 and with over 50 billion total app downloads since the first Android phone was released in 2008, cyber criminals naturally expanded their malicious activities against Googles mobile platform. Mobile threat researchers indeed recognise an alarming increase of Android malware from 2012 to 2013 and estimate that the number of detected malicious apps is in the range of 120.000 to 718.000 [3–5]. To efficiently detect malware from applications available on official and third-party sources, many efforts have contributed to studying the nature of smartphone platforms and their applications in the past decade. Google tests apps for possibly malicious behaviour through a service called Bouncer [6]. Bouncer examines apps submitted to the Android Market automatically by execution inside a virtual Android environment in Googles cloud infrastructure. Although malware download numbers decreased since the installation of Bouncer, this system does not provide security against modern attack approaches [7]. The Android platform employs

the permission system to restrict applications privileges to secure the sensitive resources of the users [8]. An application needs to get a users approval for the requested permissions to access the privacy-relevant resources. Thus, the permission system was designed to protect users from applications with invasive behaviours, but its effectiveness highly depends on the users comprehension of permission approval.

The developer is responsible for determining appropriately which permissions an application requires. According to [9, 10] and [11], most users do not understand what each permission means and blindly grant them, allowing the application to access sensitive information of the user.Another flaw is that the user cannot decide to grant single permissions, while denying others. Many users, although an app might request a suspicious permission among many seemingly legitimate permissions, will still confirm the installation.

Most of permissions defined by Google are coarse-grained. Especially, the INTERNET permission [12], the READ_PHONE_STATE permission [13], and the WRITE_SETTINGS permission [14] are coarse-grained as they give to an application arbitrary access to certain resources. The INTERNET permission allows an application to send HTTP(S) requests to all domains, and connect to arbitrary destinations and ports [15]. As a result, the INTERNET permission provides insufficient expressiveness to enforce control over the Internet accesses of the application [12]. Because of the previous problems, researchers have been involved to determine mechanisms that employs individual permissions and the combination of permissions to detect and characterise malwares. As such, Enck *et al.* [16] introduces a policy based system called Kirin to detect malwares at install time based on undesirable combination of permissions. Many works evaluate the detection of malwares with permissions using machine learning on Android [17–21]. They all realise that a permission-based mechanism can be used as a quick filter to identify malicious applications. Zhou and Jiang [22] characterise Android applications (both normal and malware applications) with individual permissions focusing on the number of occurrences of permissions in those groups.

Given the large number of published researches on Android security, especially on Android permission framework, the present work gives the current state of mechanisms for detection and classification of the malwares. In particular, we identify what works and their limitations. Finally, we illustrate promising aspects for future research.

2. BACKGROUND

The Android security model is based mainly on permissions. A permission is a restriction limiting access to a part of the code or to data on the device. The limitation is imposed to protect critical data and code that could be misused to distort or damage the user experience [9]. Permissions are used to allow or restrict application access to restricted APIs and resources. For example, the Android INTERNET permission is required by apps to perform network communications; so, opening a network connection is restricted by the INTERNET permission. An application must have the READ_CONTACTS permission in order to read entries in a users phonebook as well. To require a permission, the developer specifies using the Manifest file in declaring a <uses-permission> attribute. The android:name field specifies the name of the permission (**Figure 1**).

For example, if an app wants to communicate over the network, it would need to have an entry like what it is shown in fig2.

Natively, Android Operating System defines about several permissions which are specified as static

```
<uses-permission android:name="string" />
```

Figure 1. Declaration of permission.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.testapps.test1">
...
<uses-permission android:name="android.permission.INTERNET" />
...
</manifest>
```

Figure 2. Permissions part of the manifest file.

string members in the android.Manifest.permission class. **Table 1** gives some examples of permissions.

Table 1. Example of permissions.

Manifest Permission Strings	Description
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed.
MODIFY_PHONE_STATE	Allows modification of the telephony state such as power on
WRITE_SMS	Allows an application to write SMS messages.
READ_CONTACTS	Allows an application to read the users contacts data.

A permission can be associated with one of the following four protection levels [23]:

- Normal: A low-risk permission which allows applications to access API calls (e.g., SET_WALLPAPER) causing no harm to users.
- Dangerous: A high-risk permission which allows applications to access potential harmful API calls (e.g., READ_CONTACTS) such as leaking private user data or control over smartphone device. Dangerous permissions are explicitly shown to the user before an app is installed and the user must choose to grant the permissions or not, determining whether the installation continues or fails, respectively.
- Signature: A permission which is granted if its requesting application is signed with the same certificate as the application which defines the permission is signed.
- Signature-or-system: A permission which is granted only if its requesting application is in the same Android system image or is signed with the same certificate as the application which defines the permission is signed.

An application can additively define its own custom permissions to protect its resources, by declaring a <permissions/> attribute. It can also specify permissions to restrict access to the system components by defining the android: permission. The following example illustrates a custom permission.

```
<manifestxmlns:android="http://schemas.android.com/apk/res/android"
                package="com.example.testapps.test1">
...
<permission android: name="com.example.testapps.test1.perm.READ_INCOMING_EMAIL"
android: label= "Read incoming email"
android: description="description of the permission "
android: protectionLevel="dangerous"
android: permission ="android.permission-group.PERSONAL_INFO"
 />
 ...
</manifest>
```

This block of XML defines a new permission, READ_INCOMING_EMAIL belonging to the com.example.testapps.test1 package. The attribute protectionLevel is set to Dangerous, consequently this permission will be displayed to the user. In this case, the new permission is of type PERSONAL_INFO defined in the android.permission-group package, so it will be presented to the user right next to other permissions that allow access to personal information when the list is presented to the user.

At install-time, a user is shown with a list of permissions which an application requests as shown in **Figure 3**. The user must either grant or deny all of these permissions together. After the user approves the permission request and installs the application, the application owns its permissions throughout its lifetime and it does not need to request them again at run-time. Android controls Inter-Component Communication (ICC) through a reference monitor. The reference monitor provides a Mandatory Access Control (MAC) enforcement on how applications access components by evaluating whether the applications are granted with necessary permissions.

3. PERMISSION-BASED MALWARE DETECTION MECHANISMS

This section discusses research related to permission based malware detection mechanisms into works that analyse permissions to make decisions, those characterizing applications using individually permission semantics, association of permission semantics, and machine learning techniques.

3.1 Permission Analysis

This section presents works that analyse the permissions requested to make decisions on the behalf of the user. Holavanalli *et al.* [24] propose Flow Permissions, an extension to the Android permission mechanism. They allow users to examine and grant explicit information flows within an application as well as implicit information flows across multiple applications. VetDroid [25] is a dynamic analysis platform for reconstructing sensitive behaviour in Android apps from the permission use behaviour i.e. how applications use permissions to access sensitive system resources, and how these acquired permissions sensitive resources are further utilised by the application. Felt *et al.* [15] evaluated whether permissions of an application are effective at protecting users after collecting the permission requirements of a large set of Google Chrome extension and Android applications. Their results indicated that application permissions can have a positive impact on system security when applications permission requirements are declared upfront by the developer, but can be improved. However, this study shows that users are frequently presented with requests for dangerous permissions during application installation in install time systems. As a consequence, installation security warnings may not be an effective malware prevention tool, even for alerting users. The previous work was extended by [26] and provides guidelines for platform designers

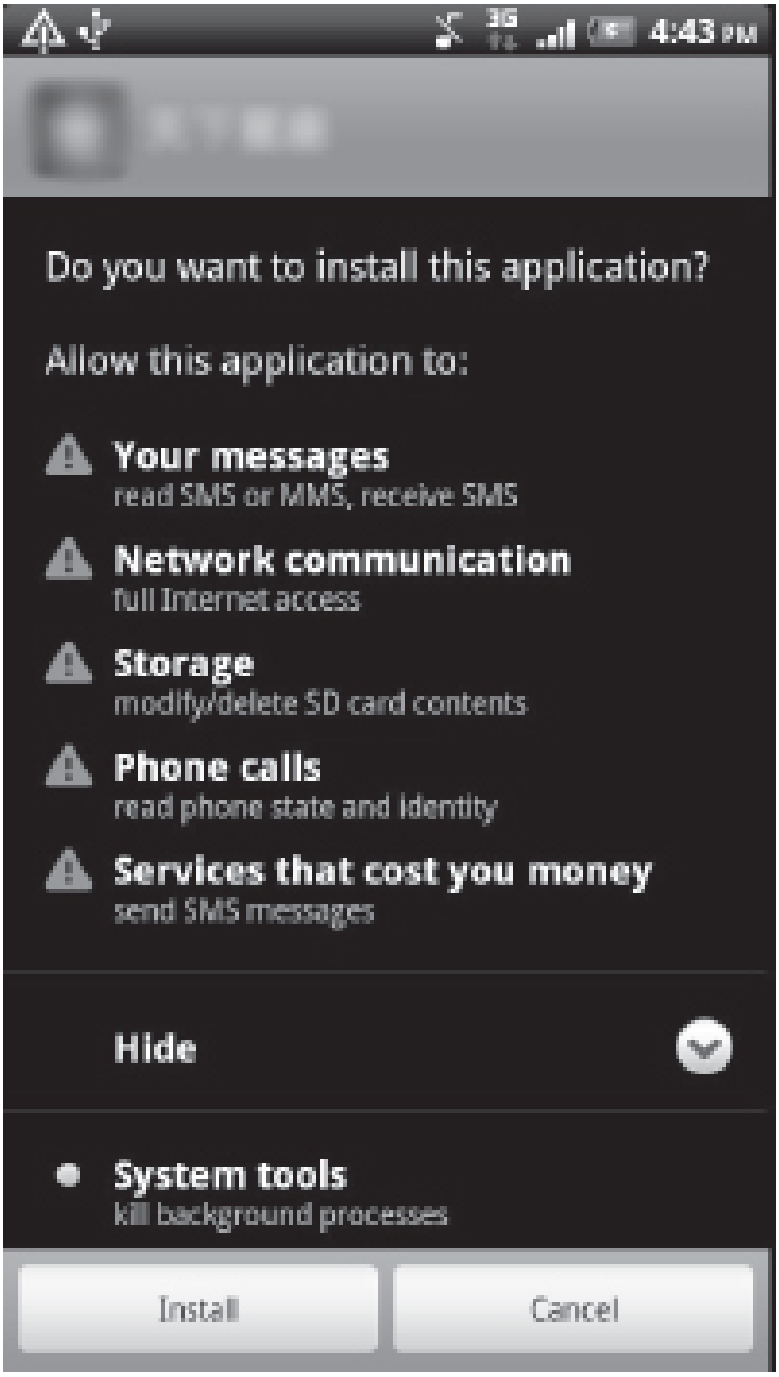

Figure 3. Permissions part of the manifest file.

in determining the most appropriate permission granting mechanism for a given permission. Permission-based security rules were used by Kirin to design a lightweight certification framework that could mitigate malware at install time. Rosen *et al.* [27] present an approach to provide to the users the necessary knowledge to make informed decisions about the applications they install, using mappings between API calls and fine-grained privacy-related behaviours. Barrera *et al.* [12] perform an empirical analysis on the expressiveness of Androids permission sets and discusses some potential improvements for Androids

permission model. Their work is based on signature verification, UID assignment and how they relate to granting of permissions. Grace *et al.* [28] detect mechanisms by which permissions granted to one application can be leaked to another, either inadvertently or deliberately through collusion. Woodpecker, is a tool that examines how the Android permission based security model is enforced in preinstalled apps and stock smartphones. PermissionWatcher [29] is an Android application that analyses permissions of other applications installed on the phone. Based on a custom set of rules, it classifies applications as suspicious if any rules apply. Through a home screen widget PermissionWatcher increases users awareness of potentially harmful applications. Sarma *et al.* [30] investigate the feasibility of using both the permissions requested by an app, the category of the application and what permissions are requested by other applications in the same category to better inform users whether the risks of installing an app is commensurate with its expected benefit. Dini *et al.* [31] propose a multi-criteria evaluation of Android applications, to help the user to easily understand the trustworthiness degree of an application, both from a security and a functional side. They assign to each permission a threat score according to the criticality of both resources and critical operations they control; they compute a global threat score for each application, which is a function of the threat score of all the required permissions. Di Cerbo *et al.* [32] describe a methodology for the detection of malicious applications in a forensics analysis. Malicious applications in this context are those that have access capabilities to sensible data and transmission capabilities as well, at the same time deceiving the users, by pretending to offer services that typically do not require such capabilities, or to make a legitimate use of them. The methodology relies on the comparison of the Android security permission of each application, with a set of reference models, for applications that manage sensitive data.

3.2 Characterisation and Classification of Applications

Another way is that given a dataset of applications, how to classify the benign and the normal ones. Here it is described techniques following this idea with the concept of permissions. It is subgrouped in approaches based on permission individually, combination of permissions, and machine learning techniques.

3.2.1 Permission individually

Zhou and Jiang [22] characterise existing Android malware from various aspects, including the permissions requested. They identified individually the permissions that are widely requested in both malicious and benign apps. According to this work, malicious apps clearly tend to request more frequently on the SMS-related permissions, such as READ_SMS, WRITE_SMS, RECEIVE_SMS, and SEND_SMS. The result is the same with RECEIVE_BOOT_COMPLETED and CHANGE_WIFI_STATE permissions.They found that malicious apps tend to request more permissions than benign ones. Barrera *et al.* [12] found no strong correlation between applications categories and requested permissions, and introduce a self-organising method to visualise permissions usage in different app categories. Sanz *et al.* [33] proposed a method for categorising Android applications through machine-learning techniques. Their method extracts different feature sets: (i) the frequency of occurrence of the printable strings, (ii) the different permissions of the application itself and (iii) the permissions of the application extracted from the Android Market. The aim of their work is to classify Android applications into several categories such as entertainment, society, tools, and productivity, multimedia and video, communication, puzzle and brain games. Orthacker

et al. [34] develop a method that circumvents the permission system by spreading permissions over two or more apps that communicate with each other via arbitrary communication channels. Sato *et al.* [35] is a method that analyses manifest files in Android application by extracting four types of keyword lists: (1) Permission, (2) Intent filter (action), (3) Intent filter (category), and (4) Process name. This approach determines the malignancy score by classifying individually permissions as malicious or benign.

3.2.2 Combination of Permissions

DroidRanger [36] is a system that characterise and detect a large set of Android malwares integrates with two schemes: The first one is Permission-based behavioural footprinting to detect the infection based on essential group of permissions requested by families of malwares and also match applications to malware information in manifest, semantics in the byte code and structural layout of the application; the second one is heuristics-based filtering scheme that defines suspicious behaviours from possibly malicious apps and then uses them to detect suspect apps. The footprint engine filters applications. PermissionWatcher [29] helps classifying Android applications being a potential threat to the user based on a set of rules and attack scenarios. They correlate resources to access to permission combinations and derive possible rules and attack. Rassameeroj and Tanahashi [37] performed a high level contextual analysis and an exploration of Android applications based on their implementation of permission based security models by applying network virtualisation techniques and clustering algorithms. From that, they discovered new potentials in permission based security models that may provide additional security to the users. This method axed on network classification helps to define irregular permission combinations requeted by abnormal applications. Gomez and Neamtiu [38] classify malicious apps into four classes of malwares: DroidDream, DroidDreamLight, Zsone SMS, Geinimi. This categorisation is based on resources accessed by these four families, the infiltration technique and the payload used. Wei *et al.* [39] present the nature, sources and implications of sensitive data on Android devices in enterprise settings. They characterised malicious apps and the risks they pose to enterprises. Finally, they have proposed several approaches for defending against security risks for enterprise. From the analysis of third-party applications, Permission additions dominate the evolution of third-party apps, of which Dangerous permissions tend to account for most of the changes. Tang *et al.* [40] introduce an extending of Android Security Enforcement with a Security Distance Model to mitigate malware. A Security Distance Pair is the quantitative representation of the security threat that this pair of permissions may cause. A permission pairs security distance is consisting of a threat point which represents the danger level and related characters. Canfora *et al.* [41] propose a method for detecting malware based on three metrics, which evaluate: the occurrences of a specific subset of system calls, a weighted sum of a subset of permissions that the application required, and a set of combinations of permissions. Enck *et al.* [16] introduce a policy based system called Kirin to detect malwares at install time based on undesirable combination of permissions. Su and Chang [42] determine whether an application is a malware depending on the set of permissions requested and announced in the manifest. They compute a score depending on the number of occurrences of each of the permissions in malwares and goodwares. Compared with previous work ([18, 20, 43]), [19] takes into account the frequency of occurrences of two permissions as a pair. The occurrence of two permissions as a pair in one app can reflect the apps potential malicious activities in some aspects, such as an app with RECEIVE_SMS and WRITE_SMS can hide or tamper with incoming SMS messages [44]. Zhu *et al.* [45] proposed a permission-based abnormal application detection framework which identifies potentially dangerous applications by the reliability of their permission lists and the applications description.

3.2.3 Machine Learning techniques

Sanz *et al.* [20] introduce a new method to detect malicious Android applications through machine learning techniques by analysing the extracted permissions from the application itself. Features used to classify are the presence of tags uses-permission and uses-feature into the manifest as well as the number of permissions of each application. MAMA [21] is a method that extracts several features from the Android manifest of the applications to build Machine Learning classifiers and detect malwares. These features are the permission requested individually and the <<uses-feature>> tag. Huang *et al.* [18] explore the possibility of detection malicious Android applications based on permissions and 20 features from Android application packages. Their experiments show that a single classifier is able to detect about 81% of malicious applications. According to them, by combining results from various classifiers, it can be a quick filter to identify more suspicious applications. Aung and Zaw [46] propose a framework that intends to develop a machine learning-based malware detection system on Android to detect malware applications and to enhance security and privacy of Smartphone users. This system monitors various permission based features and events obtained from the android applications, and analyses these features by using machine learning classifiers to classify whether the application is benign or malware. Shabtai *et al.* [47] classifies two types of Android applications: tools and games. For their point of view, successful differentiation between games and tools is expected to provide positive indication about the ability of such methods to learn and model Android benign applications and potentially detect malware files with Machine Learning (ML) techniques on static features that are extracted from Androids application files. DREBIN [17] uses permissions required and requested combined to six other features from the manifest and the dissambled code. They apply machine learning techniques for automatically learning a separation between malicious and benign applications. Once, the Support Vector Machine trained offline on a dedicated system and only it is transfered the learned model to the smartphone for detecting malicious applications. Similarly, Liu and Liu [19] employ requested permissions and required permissions by an application. In this scheme, the machine learning techniques and permissions are used to classify an app as benign or malicious.

4. ANALYSIS OF PERMISSION-BASED MALWARE DETECTION MECHANISMS

Here, we discuss the limitations of previous approaches and gives insight enhancements for future research. Permissions are one of the key for the security on Android. Previous works evaluate the detection of malwares with permissions. They all realise that a permission-based mechanism can be used as a quick filter to identify malicious applications, and that it required to be associated with a second element (such as dynamic analysis) to make complete analysis to a reported malicious application. This result is justified with interesting performance indicators such as true positive, false positive, true negative and false negative. For the works on permission analysis, solutions provided help the user to make choice of permissions, while either extending the installation system (without the participation of the user) or presenting an interface with necessary information on permissions (with the participation of the user).

4.1 Limitations

In these works, the authors restrict the study to the most requested permissions (or a precise set of permissions) [35]. However, other permissions such as READ_LOGS can be as malicious as the others (INTERNET, for example) depending on the attack. Every permission should be prudently considered as potentially risky once combined with another one.

Researchers on Android require machine learning techniques for classification (such as Decision Tree Classifiers) between benign and malicious applications. While learning techniques provide a powerful tool for automatically inferring models, they require a representative basis of data for training. The quality of the detection model of such systems critically depends on the availability of representative malicious and benign applications [17]. While it is straightforward to collect benign applications, gathering recent malware samples requires some technical effort. Another limitation of the use of machine learning is the possibility of mimicry and poisoning attacks [48, 49]. Obfuscation strategies, such as repackaging, code reordering or junk code insertion renaming of activities and components between the learning and detection phase may do not affect the model [50, 51]. An attacker may succeed in lowering the detection score by incorporating benign features or fake invariants into malicious applications [48]. The selection of the right classifier is somehow difficult depending on the classes of applications to be trained with. As a consequence, the rate of false negatives errors arise in the process of filtering. Also, if too many features are used in machine learning, detection rate may be decreased. As it is done in most of the previous works, authors simply select a random set of known malware without the consideration of the history to train a malware detector. According to [52], this way of selection yields significantly biased results. Machine learning-based detection approaches are known to present two limitations: they have high false alarm rates and determining what features should be learned in the training phase is a complex endeavour. A key step in these approaches thus resides in the process of selecting datasets for training. The performance degrades over time: for a given month Mi whose applications have been used for the training datasets, the obtained classifier is less and less able to identify all malware in the following months Mk, k>i.

The number of features to extract from the manifest (as in [41] and [18]) increases the computing overhead and the inefficiency of the solution. The choice of the feature to associate is relevant, because its modification can give false results. For instance, Zhu *et al.* [45] give acceptable results if the description is really filled by the developer. If it not the case, the outputs will be false. This is also the case for [38] which is inadequate for detecting unknown malwares because applications are classified using characteristics of malwares families: [38, 39, 47] classify applications specifically to known families of malwares, categories and enterprise ecosystem.

Most of these works extract a feature set to represent the applications. The information carried by those features is different from work to work. There is no evidence to show which features give the best detection result, but required permissions are considered in each study. Moonsamy *et al.* [53] are interested in taking the permissions as the only feature to represent the applications and find specific permission patterns to show the difference between clean and malicious applications.

The problem of usability of solutions remains urgent for the security on Android [54]. Many security solutions (Flowdroid [55], Comdroid [56]) for Android are harder to install even for expert users. The deployment is often not applicable in real devices, requiring to install components by command line. This fact let users being exposed. The objective of security systems based on permissions is to help users to evacuate malicious applications by clearly understanding interfaces. Most of the approaches using machine learning classifiers proposed are just theoretical: there is no application built to validate the

results found. This shows eventually the difficulty of practicability of such mechanisms.

Using machine learning techniques and permissions to classify an application as malicious or benign, only examines the data available to the user before an application is downloaded, the source code of the applications is not be taken into consideration. This means that implicit vulnerabilities, like permission escalation attacks and capability leaks (well described in [57]) cannot be detected.

4.2 Enhancements

Some efforts should be made to improve the effectiveness of permission-based solutions.

For reason of completeness, unlike previous works one should consider not only the 130 official permissions in Android, but also additional ones published in the GitHub [1] of Android and third-party permissions not listed in previous sources. The reason is to consider every permission as risky when it is combined to others. The research should study all these permissions rather to focus on most requested one.

For reasons of flexibility and performance, security system should learn itself from application profiles rather to use machine learning techniques. In case that machine learning techniques are used, Applications, including malware, used for training in machine learning-based malware detection must be historically close to the target dataset that is tested. Older training datasets indeed cannot account for all malware lineages, and newer datasets do not contain enough representatives of most malware from the past. as stated by Allix *et al.* [52]. Building a reliable training dataset is essential to obtain the best real-world performance. There are methods for offline analysis, such as DroidRanger [36], AppsPlayground [50] and RiskRanker [58], that might help to automatically acquire malware and provide the basis for updating and maintaining a representative dataset for such techniques over time.

One should avoid using several features to construct the vector that represents an application. This could increase significant overhead. Some approaches such as [41] for categorisation Android applications start by identifying (often intuitively) set of permissions as features for detection. It is rather more straightforward to determine after careful considerations different clusters of permissions to categorise applications in normal and malicious applications. As suggested by [59] that suffers from clear distinction, the refinement is effective with the use of weight on permissions based on its frequency.

None of previous works that determine occurrences of permissions examine duplicated permissions in the manifest. For the reason of precision, the extraction of permissions from applications considers this possibility. The percentage of permission occurrence in malwares and goodwares is one of the features often used by works aiming to characterise malwares based on permissions. They consider the gap of use permissions between normal applications and malware, and the gap that tends to zero is not signified. The best approach should find a correlation to consider every proportion of use of permissions in malwares and goodwares. Even when a permission is present just once in the malware, it must be significantly noted. Comparatively to other works on permission based detection ([42], for instance), apart from applying training and testing sets, unknown samples of applications must be collected (most recent ones historically according to [52]) to evaluate and validate the approach.

[1] https://github.com/android/platform_frameworks_base

Further, it is recommended to implement a usable application related to model, that the user could install. A survey on the usage can be performed to evaluate the usability in order to improve the design accordingly. Also the

For the detection of implicit vulnerabilities, one should associate a dynamic module that learns behaviour of application that cannot be detected based on permission profile [60]. A dynamic analysis of system calls and its parameters can be exploited in this case.

5. RELATED WORK

At the time of writing, there is no work that investigates in the literature only mechanisms for the characterisation and detection of malwares based on permissions with different techniques: individually usage of permissions, combination of permissions and machine-learning classifiers. Authors rather review Android permission system. Fang *et al.* [57] investigate the arising issues in Android security including coarse granularity of permissions, incompetent permission, Permission based security administration, insufficient permission documentation, over-claim of permissions, Access control permission escalation attack, and TOCTOU (Time of Check to Time of Use) attack. They investigate the existing counter-measures to address these issues and propose several methods to further mitigate the risk in Android security.

Egners *et al.* [61] detail the permission model of Android (similarly to [62]) and presented a selection of attacks that can be composed to fully compromise a users device using inconspicuously looking applications requesting non suspicious permissions. Wei *et al.* [63] describesthe permission evolution and usage of the Android system including the platform, third-party apps, and pre-installed apps. They found that the set of permissions tends to grow, and the growth is not aimed towards providing finer-grained permissions but rather towards offering access to new hardware features. Particularly, the set of Dangerous permissions is increasing. Android third-party and pre-installed apps do not follow the least privilege principle because of the presence of over-privilege in these apps and the fact that apps tend to use more permissions over the time. One important note is that pre-installed apps use higher-privileged permissions which pose security and privacy risks. Reddy *et al.* [64] introduced the idea of application-centric permissions and argued that they are an expressive and practical approach to increase the security of Android apps. Enck [60] reviews the literature of smartphone research in general, including efforts in designing new OS protection mechanisms, as well as performing security analysis of real applications. As the present study that works only on Android, Enck analyses mechanisms based on permission analysis and indicate that permissions are valuable for performance efficient security analysis but considering permissions alone is limited, and they are likely best used to steer dynamic and static analysis. Compared to Enck's work, our work specify precisely directions to improve permission based approaches on Android, rather to discuss several additional areas for future smartphone security research.

6. CONCLUSION

Smartphone security research is growing in popularity. To help future research, this study describes the model of Android permissions, investigates existing permissions-based approaches (with individual usage of permissions, combination of permissions and machine learning techniques) to enhance security, and discusses their advantages and limitations. Finally, several considerations have been discussed to improve

these mechanisms for future Android security research in these aspects.

References

[1] "Canalys: Over 1 billion Android-based smart phones to ship in 2017," 2013. http://www.canalys.com/newsroom/over-1-billion-android-based-smart-phones-ship-2017.

[2] B. Wire, "Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC," 2013.

[3] "Juniper Networks: Third Annual Mobile Threats Report," 2013. http://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2012-mobile-threats-report.pdf.

[4] "Kindsight: Security Labs Malware Report - Q2 2013," 2013. http://www.kindsight.net/sites/default/files/Kindsight-Q2-2013-Malware-Report.pdf.

[5] T. Micro, "TrendLabs 2Q 2013 Security Roundup," 2013.

[6] H. Lockheimer, "Android and security," *Google Mobile Blog, Feb*, vol. 2, 2012.

[7] X. Jiang, "An evaluation of the application (app) verification service in android 4.2, January 2014."

[8] M. Frank, B. Dong, A. P. Felt, and D. Song, "Mining Permission Request Patterns from Android and Facebook Applications," in *ICDM*, pp. 870–875, 2012.

[9] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, p. 3, ACM, 2012.

[10] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A conundrum of permissions: installing applications on an android smartphone," in *Financial Cryptography and Data Security*, pp. 68–79, Springer, 2012.

[11] F. Tchakounté, P. Dayang, J. M. Nlong, and N. Check, "Understanding of the Behaviour of Android Smartphone Users in Cameroon: Application of the Security,"

[12] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 73–84, ACM, 2010.

[13] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pp. 71–72, ACM, 2012.

[14] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster, and T. Millstein, "Dr. Android and Mr. Hide: Fine-grained security policies on unmodified Android," 2011.

[15] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of install-time permission systems for third-party applications," tech. rep., Technical report, University of California at Berkeley, 2010. UCB/EECS-2010-143, 2010.

[16] W. Enck, M. Ongtang, and P. McDaniel, "Mitigating Android software misuse before it happens," 2008.

[17] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," 2014.

[18] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," in *Advances in Intelligent Systems and Applications-Volume 2*, pp. 111–120, Springer, 2013.

[19] X. Liu and J. Liu, "A Two-Layered Permission-Based Android Malware Detection Scheme," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pp. 142–148, IEEE, 2014.

[20] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pp. 289–298, Springer, 2013.

[21] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Álvarez Marañón, "MAMA: Manifest Analysis for Malware Detection in Android," *Cybernetics and Systems*, vol. 44, no. 6-7, pp. 469–488, 2013.

[22] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 95–109, IEEE, 2012.

[23] "Google: Android permissions," 2013. http://developer.android.com/guide/topics/manifest/permission-element.html.

[24] S. Holavanalli, D. Manuel, V. Nanjundaswamy, B. Rosenberg, F. Shen, S. Y. Ko, and L. Ziarek, "Flow permissions for android," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pp. 652–657, IEEE, 2013.

[25] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 611–622, ACM, 2013.

[26] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, D. Wagner, *et al.*, "How to Ask for Permission," in *HotSec*, 2012.

[27] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," in *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 221–232, ACM, 2013.

[28] M. C. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *NDSS*, 2012.

[29] E. Struse, J. Seifert, S. Üllenbeck, E. Rukzio, and C. Wolf, "PermissionWatcher: Creating User Awareness of Application Permissions in Mobile Systems," in *Ambient Intelligence*, pp. 65–80, Springer, 2012.

[30] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: a perspective combining risks and benefits," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pp. 13–22, ACM, 2012.

[31] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "A Multi-Criteria-Based Evaluation of Android Applications," in *Trusted Systems*, pp. 67–82, Springer, 2012.

[32] F. Di Cerbo, A. Girardello, F. Michahelles, and S. Voronkova, "Detection of malicious applications on android os," in *Computational Forensics*, pp. 138–149, Springer, 2011.

[33] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas, "On the automatic categorisation of android applications," in *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pp. 149–153, IEEE, 2012.

[34] C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber, "Android security permissions–can we trust them?," in *Security and Privacy in Mobile Information and Communication Systems*, pp. 40–51, Springer, 2012.

[35] R. Sato, D. Chiba, and S. Goto, "Detecting Android Malware by Analyzing Manifest Files," *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, pp. 23–31, 2013.

[36] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets.," in *NDSS*, 2012.

[37] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: evaluating android anti-malware against trans-

formation attacks," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 329–334, ACM, 2013.

[38] L. Gomez and I. Neamtiu, "A Characterization of Malicious Android Applications," tech. rep., Technical report, University of California, Riverside, 2011.

[39] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Malicious android applications in the enterprise: What do they do and how do we fix it?," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pp. 251–254, IEEE, 2012.

[40] W. Tang, G. Jin, J. He, and X. Jiang, "Extending Android security enforcement with a security distance model," in *Internet Technology and Applications (iTAP), 2011 International Conference on*, pp. 1–4, IEEE, 2011.

[41] G. Canfora, F. Mercaldo, and C. A. Visaggio, "A classifier of malicious android applications," in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pp. 607–614, IEEE, 2013.

[42] M.-Y. Su and W.-C. Chang, "Permission-based malware detection mechanisms for smart phones," in *Information Networking (ICOIN), 2014 International Conference on*, pp. 449–452, IEEE, 2014.

[43] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using bayesian classification," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pp. 121–128, IEEE, 2013.

[44] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 235–245, ACM, 2009.

[45] J. Zhu, Z. Guan, Y. Yang, L. Yu, H. Sun, and Z. Chen, "Permission-based abnormal application detection for android," in *Information and Communications Security*, pp. 228–239, Springer, 2012.

[46] Z. Aung and W. Zaw, "Permission-based android malware detection," *International Journal Of Scientific & Technology Research*, vol. 2, no. 3, 2013.

[47] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying Android applications using machine learning," in *Computational Intelligence and Security (CIS), 2010 International Conference on*, pp. 329–333, IEEE, 2010.

[48] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading worm signature generators using deliberate noise injection," in *Security and Privacy, 2006 IEEE Symposium on*, pp. 15–pp, IEEE, 2006.

[49] S. Venkataraman, A. Blum, and D. Song, "Limits of learning-based signature generation with adversaries," 2008.

[50] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: automatic security analysis of smartphone applications," in *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 209–220, ACM, 2013.

[51] M. Zheng, P. P. Lee, and J. C. Lui, "ADAM: an automatic and extensible platform to stress test android anti-virus systems," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 82–101, Springer, 2013.

[52] K. Allix, T. F. D. A. Bissyande, J. Klein, and Y. Le Traon, "Machine Learning-Based Malware Detection for Android Applications: History Matters!," 2014.

[53] V. Moonsamy, J. Rong, and S. Liu, "Mining permission patterns for contrasting clean and malicious android applications," *Future Generation Computer Systems*, vol. 36, pp. 122–132, 2014.

[54] F. Tchakounté and P. Dayang, "Qualitative Evaluation of Security Tools for Android," *International Journal of Science and Technology*, vol. 2, no. 11, 2013.

[55] C. Fritz, S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. le Traon, D. Octeau, and P. McDaniel, "Highly precise taint analysis for Android applications," *EC SPRIDE, TU Darmstadt, Tech. Rep*,

2013.

[56] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *Proceedings of the 9th International Conference on Mobile systems, applications, and services*, pp. 239–252, ACM, 2011.

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

[58] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 281–294, ACM, 2012.

[59] I. Rassameeroj and Y. Tanahashi, "Various approaches in analyzing android applications with its permission-based security models," in *Electro/Information Technology (EIT), 2011 IEEE International Conference on*, pp. 1–6, IEEE, 2011.

[60] W. Enck, "Defending users against smartphone apps: Techniques and future directions," in *Information Systems Security*, pp. 49–70, Springer, 2011.

[61] A. Egners, U. Meyer, and B. Marschollek, "Messing with Android's Permission Model," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pp. 505–514, IEEE, 2012.

[62] W. Enck, M. Ongtang, P. D. McDaniel, *et al.*, "Understanding Android Security," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 50–57, 2009.

[63] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 31–40, ACM, 2012.

[64] N. Reddy, J. Jeon, J. Vaughan, T. Millstein, and J. Foster, "Application-centric security policies on unmodified Android," *UCLA Computer Science Department, Tech. Rep*, vol. 110017, 2011.