DEMYSTIFYING ANDROID: A REVIEW OF RECENT RESEARCH

Sangeeta Rani
Mata Gujri College, Fatehgarh Sahib, Punjab

ABSTRACT-Smartphone, with its powerful capabilities, has been wildly used in every area of our day to day life. The enormous kinds of applications installed in these smart phones like WhatsApp and Viber have changed the way people live and communicate. An estimate by Gartner indicates that 90% of the phones by the end of 2018 will be smartphones. The most popular mobile operating system today in the industry is Android. However, with the prevalence of Android smartphone, the malware writers have begun to target it. Android use permission- based system to warn the users about the risk associated with an application. This paper will review the growth of malware on Android platform to provide an understanding of the real risk and behavior of these types of threats. We will also review some countermeasures and tools to analyze Android applications which are based on Android Permission Model. The findings of the review will be making an effort to research in the area of Android malware detection by understanding the current scenario and developments that have happened in the field thus far.

Keywords- Android, Apps, Malware, Permissions, Smartphone, Permission Model.

1. INTRODUCTION

Smartphones have become an integral part of almost everyone's daily life. As the capabilities and functionality of these devices are constantly improving and new innovative technologies are emerging, more and more people rely on smartphones technologies. Nearly every small scale mobile device includes features like high-bandwidth, web access, location tracking, ability to access digital media etc. The amount and importance of personal data carried by smartphones have become similar to that stored in personal computers and laptops. According to a prediction of IMS Research (the topmost independent market research and consultancy in the global electronics industry), the annual smartphone sales will exceed 1 billion devices at the end of 2016[1]. All the sophisticated capabilities and data in modern mobile phones make them very attractive to malicious software developers, and increase the probability of their use in various illegal activities. Smartphones need to run lightweight operating system due to limited battery and storage constraints. Android has become the most popular operating system with 79 percent of smartphone market share in 2017[2].

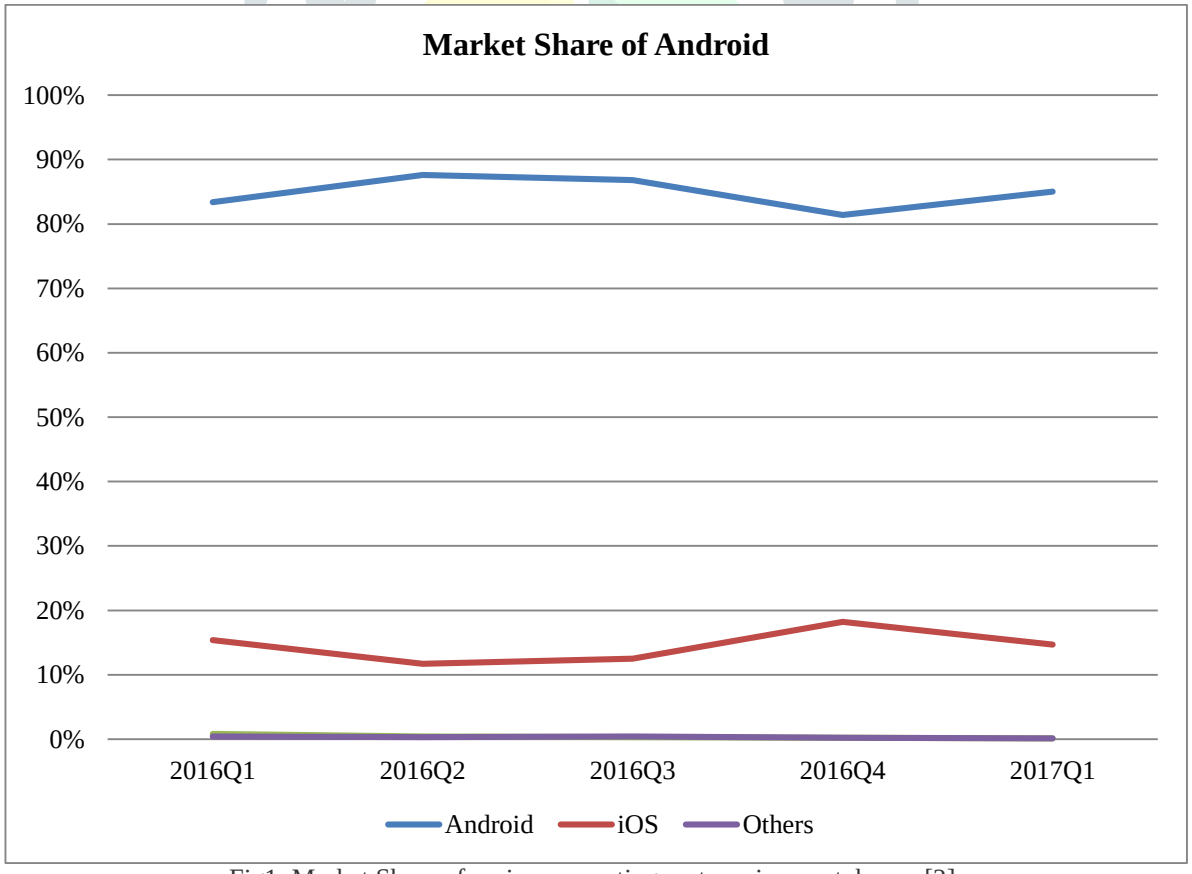

Fig1: Market Share of various operating systems in smartphones [2].

In Android users can download applications from official Google Play Store, but it allows users to install applications from third party application stores also. With a current number of 2.8 million apps available(in 2017) for download via Google's official Play Store [3], and possibly another million available on third party application stores, we can estimate that there are over 20,000 new applications being released every month. The major motivation behind the malware attacks on android is the popularity of Android. According to F-secure security report 2017, 99% of the total malware in the market is designed for Android [4]. This has the prime reason for recent research on both malware attacks and protection level of Android devices. Given a large number of published researches on Android security, especially on Android permission model, we provide a systematic overview of the current state of Android security. In particular, we investigate the recent advancement on Android security, identify the issues in Android permission model, and analyze the countermeasures to address the security issues. The rest of this paper is organized as follows: Section 2 introduces the Android Permission Model. Section 3 discusses the issues in Android permission model. Sections 4 present countermeasures and compare the tools to detect malware. Finally, Section 6 concludes the paper.

2. Android Permission Model

The security architecture of Android, the Google Chrome browser extension system and the Blackberry platform, all use a similar security model called the permission-based security model [5]. A permission-based security model can be loosely defined as a model in which

1) Each application is associated with a set of permissions that allows accessing certain resources;

2) Permissions are explicitly accepted by users during the installation process and checked at runtime when resources are requested.

Android applications files are packed in files with apk extension. These files contain all the classes and resources required by the applications [6]. The binary classes are converted into a proprietary format for Dalvik Virtual Machine (dex files). Each application runs in a separate virtual machine, having its own unique identifier (UID). Due to this, application resources are protected from other applications and the communication and data transfer between applications has a high degree of confidence. Even so, applications could communicate each other using messages, this being another source of threat. Android permissions themselves are much more granular than a typical LINUX permission system. They cover a large set of operations, including controlling the sleep state, accessing device hardware, accessing PII, and many system operations. Applications need user permissions in order to access restricted API. According to [7], application permissions fall in one of these categories:

•Normal – Granted automatically, Normal permissions don't present any risk for applications or system. Even the user is not informed when the applications are installed.

•Dangerous – These permissions, if used by malicious authors, could produce negative effects and the user must be aware of about it. If the permission request is denied by the user, then the application is not installed.

•Signature: Signature permissions are useful for restricting component access to a small set of applications trusted and controlled by the developer.

•Signature or System – These permissions are required for system applications and are granted only if the requesting application is signed by the same developer that defined the permission.

All permissions required need to be declared in application's AndroidManifest.xml file like this:

<uses-permission android:name= "android.permission.INTERNET">
</uses-permission>

INTERNET: Allows applications to open network sockets.

Also, applications can define custom permissions that can be used to access their resources.

3. Security Issues in Android's Permission Model

According to Fang et al. Android Permission Model is suffering from security issues including permission gap, permission escalation attack, the coarse granularity of permissions, incomplete permission administration and insufficient permission documentation [8].

Both the developers and end users usually lack in permission knowledge result in imperfect permission administration. Felt et al. [9] examined whether the Android permission system is effective at warning users. In particular, they evaluated whether during installation users pay attention to the list of required permissions displayed by the application. They performed two usability studies: an Internet survey of 308 Android users and a laboratory study wherein they interviewed and observed 25 Android users. The results displayed that only 17% of participants paid attention to permissions during installation, and only 3% of Internet survey respondents could properly answer all three permission comprehension questions. This indicates that current Android permission warnings do not help most users make correct security decisions. Further they analyzed 100 paid and 856 free Android applications, and found that most of the applications (above 80%) ask for at least one 'dangerous permission'. Changes to these permission systems are necessary to decrease the number attacks on Android platform. Barrera et al. [10] analyzed the permission distribution of many applications and revealed that most of them are of coarse granularity. According to them, the INTERNET permission is the most popular and hypothesized that most developers request this to request advertisements from remote servers. Even the internet permission allows an application to send HTTP(s) request to all domains and connect to arbitrary destinations and ports. Permission gap is also the most dangerous threat to Android Security [11]. If applications are granted more permissions than they actually need it facilitates all kinds of malicious usage. Felt et al. [12] Identified that 56% of the applications request for a large number of unnecessary permissions. Developers may make wrong decisions because of several reasons like permission name ambiguities, deprecated permissions etc. Vidas et al. [13] considered that the limited permission usage information on Android is also perplexing the developers. Privilege escalation attack [8] allows a malicious application to communicate with other applications so as to access

critical resources without requesting for corresponding permissions explicitly. An application with less permission can access components of a more privileged application because permission model allows transitive permission usage [14].

4. Permission- Based Malware Detection Mechanisms

Malware or malicious code is a short form for malicious software. It is a standalone code or malicious code attached with software which is designed with an intension to harm, steal or corrupt the data, networks or hosts [15]. Viruses, worms, rootkits, adware, trojans, spyware and botnets are all types of malware.This section present research work related to permission- based malware detection methods that analyze permissions to make decisions and various tools that detect malicious applications.

5.1 Permission Analysis

Liccardi et al. [16] collected about 528,433 apps on Google Play and analyzed the permissions requested by each app. To measure the risk factor of an app, they developed 'sensitivity score' a measure to represent the number of occurrences of permissions that read personal information about users where network communication is possible. From the dataset 46% applications collect between 1 and 20 sensitive permissions and have the ability to transmit it outside the device. The sensitivity of apps differs greatly between free and paid apps as well as between categories and content ratings. Sarma et al. [17] investigated 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 about the risks and benefits of an app. Di Cerbo et al. [18] presented a methodology in a forensic analysis to protect users from suspicious applications. 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. Rosen et al. [19] presented a static analysis approach to providing users the knowledge required to make informed decisions about the installed application. First, they produce a knowledge base of mappings between API calls and fine-grained privacy-related behaviours. They then use this knowledge base to produce high-level profiles of application behaviour. They have analyzed almost 80,000 applications and have made the resulting behaviour profiles available both through an Android application and online. Nearly 1500 users have used this application. Based on 2782 pieces of application-specific feedback, they analyze users' opinions concerning how applications affect their privacy and demonstrate that these profiles have had a considerable impact on their understanding of those applications. P. Chai in his thesis [20] investigated the permissions requested by Android applications, with the possibility of discovering malicious applications based only on information presented to the user before an application is installed. During the research work, the author collected a large data set consisting of applications available on Google Play and 3 different third-party Android application stores. These applications are analyzed using manual pattern recognition and k-means clustering, focusing on the permissions they request. The pattern analysis relies on a smaller data set consisting of confirmed malicious applications. The method is evaluated based on its ability to acknowledge malicious potential in the analyzed applications. During the analysis of this data set, they uncovered incongruous patterns in the Google Play market relating to permissions with high protection levels. Dini et al. [21] proposed a multi-criteria analysis of Android applications, to assist the user to simply understand the trustworthiness degree of an application from a security side. They allocate a threat score to each permission according to the criticality of both resources and critical operations they control; they calculate a global threat score for each application, which is a function of the threat score of all the required permissions. Felt et al. [8] use static analysis to determine whether or not an Android application is over- privileged. It classified an application as over privileged if the application requested a permission which it never actually used. With a data set of 940 applications they find that about one-third of the participants are over privileged. Their key observation was that developers are trying to follow the least privilege however generally fail due to insufficient API documentation.

Au et al.[22] survey the permission systems of many popular smartphone operating systems and taxonomies them by the amount of control they give users, the amount of information they convey to users and also the level of interactivity they need from users. Further, they discuss several issues related to extracting permissions-based information from Android applications. Peng et al. [23] used probabilistic generative models to calculate the risk in Android apps. They selected the permissions of the applications as a key feature. Specifically, they selected the top 20 most frequently requested permissions in their dataset, composed by 2 benign software collections (157,856 and 324,658 samples, respectively), downloaded from the Google Play and 378 unique samples of malware. They obtained a 0.94 area below ROC curve as a best result. Sato et al. [24] proposed a technique based on the characteristic analysis of Android manifest files and is effective for detecting well-known existing malware and unknown malware. The result shows that the new technique can effectively detect Android malware, even when the sample is unknown. The preliminary investigations discovered that there are certain differences between the manifest files of benign applications and malicious applications. Permissions typically cover broad, vague categories of functionality and provide insufficient detailed information for users to make meaningful decisions. Bartel et al. [25] presented an approach to detecting permission gaps using static analysis. Using 2 datasets of Android applications, they discovered that a non negligible part of applications suffer from permission gaps, i.e. does not use all the permissions they declare. The prototype implementation is able to automatically find 9562 Android framework entry points that check permissions. The analysis revealed that 94/742 and 35/679 downloaded applications from Android app stores indeed suffer from permission gaps. Felt et al. [26] survey the current position of mobile malware in the wild. They analyze the motivations behind forty- six pieces of iOS, Android, and Symbian malware that unfold in the wild from 2009 to 2011. They detected that four pieces of malware use root exploits to mount sophisticated attacks on Android mobiles. As a part of the survey, they use permission counts and a comparison of individual permissions to find malicious applications. Android malware usually requests the ability to directly send SMS messages, which is uncommon among benign applications. Permission-based classification would force future consideration as the set of known Android malware grows.

5.2 Tools

Felt et al. built Stowaway, a tool which detect over claim of permissions in Android applications [12]. Stowaway analyzed a collection of 940 applications and identified that about one-third of these applications request more permissions than what they need. The authors found that common unnecessary permissions include ACCESS_NETWORK_STATE, READ_PHONE_STATE, ACCESS_WIFI_STATE, WRITE_EXTERNAL_STORAGE, and CALL_PHONE. Such unnecessary permissions can be leveraged by malicious applications.

Au et al. [27] developed PScout, a static analysis tool which extracts the permission specification from the Android operating system source code. PScout overcomes many challenges, like scalability as a result of Android's 3.4 million line code base, accounting for permission enforcement across processes due to Android's use of IPC and abstracting Android's different permission checking mechanisms into a single primitive for analysis. They use PScout to analyze four versions of Android spanning version 2.2 up to the newly released Android 4.0. The study found that that while Android has over 75 permissions, there is very little redundancy in the permission specification. On the other hand, if applications could be constrained to use documented APIs, then about 22% of the non-system permissions are actually unnecessary. Permission-based security rules are used by Kirin [28] to develop a light weight certification framework that can mitigate malicious activity at install time. Apex [29] and Saint [30] are 2 extensions to the Android's permission system by introducing runtime constraints on the granted permissions.

Jeon et al. [31] presented RefineDroid, Mr. Hide, and Dr. Android, tools to understand and implement fine-grained permissions on Android platform without requiring modifications in the operating system. Using RefineDroid, they conducted a survey showing that fine-grained permissions are appropriate for a lot of popular apps. They then applied Dr. Android to transform a range of apps to use Mr. Hide. The study claims that fine- grained permissions via Dr. Android and Mr. Hide provide stronger privacy and security while affecting apps functionality and performance. Enck et al. [32] develop TaintDroid, a dynamic taint tracking and analysis system capable of tracking, multiple sources of sensitive data. TaintDroid provides control over how data flows amongst the applications. TaintDroid incurs only 14% performance overhead on a CPU-bound micro-benchmark. Using TaintDroid to observe the behaviour of thirty popular third-party Android applications, the author found 68 instances of potential misuse of users' private information across 20 applications. Fuchs et al. [33] present ScanDroid, the first static program analysis tool that enable checking of Android applications once they are installed. It detects inter-application security risks from manifest files that accompany such applications.

Zhang et al. [34] present VetDroid, a dynamic analysis platform that use the permissions to reconstruct sensitive behaviours in Android apps, i.e., how applications use permissions to access system resources, and the way these acquired resources are further used by the application. Using real-world Android malware; they show that VetDroid can clearly reconstruct fine-grained malicious behaviors to ease malware analysis. Gibler et al. [15] propose AndroidLeaks a static analysis framework for automatically finding potential leaks of private information in Android applications on a vast scale. This tool considerably decreases the number of applications and the number of traces that a security auditor needs to validate manually. The authors evaluate the efficacy of AndroidLeaks on a database of 24,350 Android applications, collected from different Android stores. AndroidLeaks found 57,299 sensitive data leaks in about 7,414 Android apps, out of which they have manually verified that 2,342 apps leak sensitive data including phone information, GPS location, WiFi data, and audio. Struse et al. [35] developed PermissionWatcher: an Android application that analyzes permissions of other applications downloaded on the phone. Based on a custom set of rules, it categorizes applications as malicious if any rules apply.

Zhang et al. [36] designed Permlyzer, a tool to automatically analyze the permission usage in Android applications. Permlyzer leverages the combination of runtime monitoring and static analysis to explore the functionality of an application. Further 51 malware families and more than 110,000 Android applications demonstrate that Permlyzer can detect permission use in all the applications. Like Permlyaer, Manilyzer, is a tool that utilizes the wealthy information in the manifest files and uses state-of-the-art machine learning algorithms to categorize an application as malicious or benign. Other than classifying applications, Manilyzer is used to study the trends of permission requests in malicious applications [37]. Sanz et al. [38] present Manifest Analysis for Malware detection in Android (MAMA), a new method that extracts several features from the Android Manifest of the applications to build machine-learning classifiers and detect malware. PMDS (Permission-based Malware Detection Systems) is a tool that automatically identify for potentially dangerous behaviour in an app based on the combination of requested permissions. It has the potential to detect unknown and zero-day malware. Experimental results demonstrate that PMDS detects more than 90% of unknown malware with a false positives rate of 1.52–3.93% [39]. Zhou et al. [40] present Droidranger for malware detection on Android stores. Applications are first sieved based on the required permissions and then filter through a heuristics-based filter. The authors use a dataset of 204,040 apps collected from 5 different Android stores. DroidRanger detects 211 malicious applications and also exposed 2 zero-day malware (in 40 apps). RiskRanker [41] also tries to uncover malware by performing a risk analysis to categorize an app as a low, medium or high risk.

5.3 Comparison of Tools

Stowaways, Pscout, RiskRanker, PMDA, Kirin, ScanDroid, and AndroidLeaks are detecting Android malware application with static analysis. Malware can be quickly detected using static analysis and it also prevents the malicious application from being installed in the device. However, the static analysis can be avoided through obfuscation or encryption technique. In contrast, the obfuscation and encryption technique is not able to evade the dynamic analysis because it can detect the behavior of the malware during runtime.

VetDroid, TaintDroid, Apex, Saint, RefineDroid, Mr. Hide are using the dynamic analysis. This approach tends to get a high generation of captured data which can consume the storage space and computational power. Besides that, times required to examine the malware activity cannot be define clearly [42].

Table 1: Comparison of Tools.

Authors	Tool	Approach	Availability	Based On	Evaluation Scale	Important Features
Felt et al.	Stowaway	Static Analysis	NA	Manifest file & API calls	940 apps	Detect over claim of permissions.
Au et al.	PScout	Static Analysis	open	API Calls	5 versions of Android OS	Detect over claim of permissions.
Enck et al.	Kirin	Static Analysis	open	App Certification	311	Mitigate malware at install time.
Nauman et al.	Apex	Dynamic	open	Modified Kernel	N/A	Introduce runtime constraints on the granted permissions.
Ongtang et al.	Saint	Dynamic Analysis	NA	Modified Kernel	N/A	Introduce runtime constraints on the granted permissions.
Jeon et al.	RefineDroid, Mr. Hide, & Dr. Android	Dynamic Analysis	NA	API calls & Manifest File	14 case study apps	Implement fine-grained permissions on Android.
Enck et al.	TaintDroid	Dynamic Analysis	open	Modified DVM & Taint Tracking	38 apps	Track the flow of sensitive data.
Fuchs et al.	ScanDroid	Static Analysis	open	Taint Tracking & Manifest File	N/A	Apply data flow analysis to check privacy leaks.
Zhang et al.	VetDroid	Dynamic Analysis	NA	Modified Kernel & TaintDroid	1249 apps	Examine the internal sensitive behaviors of an app.
Gibler et al.	AndroidLeaks	Static Analysis	NA	API Calls	24350	Detect privacy violations in Android
Struse et al.	PermissionWatcher	Static Analysis	open	Manifest File	1000+ Users	Increase users awareness about harmful apps.
Zhang et al.	Permlyzer	Hybrid	open	Call stack based	110,000	Automatically analyze the requested permissions.
Feldman et al.	Manilyzer	Hybrid	open	Manifest File(machine Learning algos)	617	Detect malicious Android applications.
Sanz et al.	MAMA	Hybrid	NA	Manifest file(Machine Learning algo's)	N/A	Detect malware
Rovelli et al.	PMDA	Static	NA	Machine learning classifiers	2950 apps	Android malware detector.
Zhou et al.	DroidRanger	Hybrid	NA	Malware behavior footprints	204,040 apps	Android malware detector.
Grace et al.	RiskRanker	Static Analysis	NA	Code path behavior	118,318 apps	Android malware detector

*NA-Not Available

Both Stopwaway and Pscout discover over claim of the permissions. Stopwaway works solely on Android version 2.2 whereas Pscout works on every version of Android [8]. As compared to Stopwaway that fuzzes the Android API directly, Pscout fuzzes the application that uses the APIs [30]. As a result, Pscout is more complex however less sound than Stopwaway. Kirin, Saint, Apex and Dr. Android and Mr. Hide are the tools that perform fine grain policy enforcement. Both Kirin and Saint perform policy enforcement at install time. Saint can also perform run- time policy enforcement. In Saint, any install time policy are defined by applications while the source of Kirin policies is mainly the author of Kirin [8]. The source of Aplex policies is mainly the end users. Saint and Apex require modifications in Android operating system; however Dr. Android and Mr. Hide run on unmodified Android systems [40]. TaintDroid detects sensitive data leaks with dynamic taint tracking. TaintDroid track only data flow, but not control flow. In Contrast, a similar tool ScanDroid uses static analysis to determine data flow through Android applications by accessing the source code. Vetdroid is a dynamic framework that constructs a permission use graph to identify actual permission use. Its data tainting method is built upon TaintDroid but improves it by identifying implicit and explicit permission use points. While TaintDroid needs manual user interaction to find data leaks [43], another tool of AndroidLeaks is completely automated. It uses static analysis approach records the event in a log which can be audited by the user if private information is leaked from the device. Premlyzer investigates the usage of permissions using hybrid approach without the goal to detect malware. However, Manilyzer examines the application for malware detection and also revealing trends in permission usage amongst malware [42]. Like manilyzer MAMA also rely on the manifest file to study the permissions and to find the malware. MAMA saves device resources by preventing malware installation instead of observing monitoring the execution of the application. PMDS, DroidRanger and RiskRanker are the tools that classify an app as malicious or benign. DroidRanger and RiskRanker rely on known malware samples. But they offer different methods to detect malware. These tools can detect zero-day malware also. Droidranger takes a heuristic-based approach whereas PMDS uses machine learning algorithms for malware detection. PMDS also tries to correlate permissions to behavior as in VetDroid.

To defend against this severe increase of Android malware and help users make a better evaluation of apps at install time, several approaches have been proposed. However, most of these solutions suffer from some shortcomings; computationally expensive, not general or not robust enough.

5. CONCLUSION

The rapid growth of mobile devices has also increased the number of mobile malware within the application stores mainly when Android OS is widely adopted in the mobile devices. During this study, we have reviewed security issues in the permission model of Android operating system. This study has also comprehensively reviewed existing Android malware and malware detection tools. We have described the various static, dynamic and hybrid tools that are used to improve the security factors and permission mechanisms like Apex, Kirin, DroidRanger etc. The Android malware detection system must take the consideration of the limited resources the mobile devices provide. This study shows that there is a need for finding a more vigorous, reliable and effective Android malware detection tools.

REFERENCES

[1] Press Releases. [Online] Available: http://imsresearch.com/press-release/Global_Smartphones_Sales_Will_Top_420_Million_Devices_in_2011_Taking_28_Percent_of_all_Handsets_According_to_IMS_Research

[2] Android climbed to 79 percent of smartphone market share in 2013, but its growth has slowed. [Online] Available: http://www.engadget.com/2014/01/29/strategy-analytics-2013-smartphone-share/

[3] Number of available Android applications. [Online] Available: http://www.appbrain.com/stats/number-of-android-apps

[4] U.S. security agencies say Android mobile main target for malware. [Online] Available: http://www.reuters.com/article/2013/08/27/net-us-android-security-dUSBRE97Q15Z20130827

[5] 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", *Proceedings of ACM Conference on Computer and Communications Security (CCS 2010)*, Chicago, Illinois, USA, October 4-8, 2010, pp. 73–84.

[6] F-secure, 'Another reason 99% of mobile malware targets Android'. [Online], Available on: https://safeandsavvy.f-secure.com/2017/02/15/another-reason-99-percent-of-mobile-malware-targets-androids/, Accessed on 11/08/2017 [7] P. Pocatilu, "Android Applications Security", Informatica Economica vol: 15 (3), pp. 163-171, 2011

[8]Z. Fang, W. Han, Y. Li, "Permission based Android security: Issues and countermeasures", Computers & Security, Elsevier Advanced technology, vol: 43, pp. 205-218, 2014.

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

[10] 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," *Proceedings of the 17th ACM conference on Computer and communications security*, ACM, New York, 2010, pp. 73–84.

[11] A. Bartel, J. Klein, M. Monperrus, Y.L.Traon, "Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges and Solutions for Analyzing Android". IEEE Trans. Software Eng. vol: 40(6), pp. 617-632, 2014.

[12] A.P. Felt , E. Chin., S. Hanna, D. Song , D. Wagner, " Android permissions demystified" ,*Proceedings of 8th ACM conference on Computer and communications security*; New York, USA. 2011b. pp. 627-638.

[13]T. Vidas, N. Christin, L. Cranor, "Curbing android permission creep," *Proceedings of the 2011 Web 2.0 Security and Privacy Workshop (W2SP 2011)*. Oakland, CA. May 2011.

[14] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," Trust and Trustworthy Computing, pp. 291–307, 2012.

[15] Malware. [Online] Available: https://en.wikipedia.org/wiki/Malware

[16] I. Liccardi, J. Pato, and D. J. Weitzner, "Improving Mobile App Selection through Transparency and Better Permission Analysis", Privacy and Confidentiality, vol: 5(2), pp. 1–55, 2013.

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

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

[19] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," *Proceedings of the 3rd ACM conference on Data and application security and privacy*, ACM, New York, USA, 2013 pp. 221–232.

[20] P. H. Chia, S. Knapskog, "Information Security on the Web and App Platforms: An Economic and Socio-Behavioral Perspective," Ph.D Thesis, Norweign university of science and technology, 2012.

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

[22] K. Au, Y. Zhou, Z. Huang, P. Gill, and D. Lie, "A look at smartphone permission models," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ACM, New York, USA.,2011, pp. 63–68.

[23] Peng, H., C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy.. "Using Probabilistic Generative Models for Ranking Risks of Android Apps," *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012. New York, USA. pp 241–252.

[24] R. Sato, D. Chiba and S. Goto1, "Detecting Android Malware by Analyzing Manifest Files," *Proceedings of the Asia-Pacific Advanced Network*, Aug 2013 vol. 36, pp. 23-31. [Online] Available: http://dx.doi.org/10.7125/APAN.36.4 ISSN 2227-3026

[25] A. Bartel, J. Klein, Y. L. Traon," Automatically securing permission-based software by reducing the attack surface: an application to Android," *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012 , pp. 274-277 .

[26] A. P. Felt, M. Finifter, E. Chin, D. Wagner, "A Survey of Mobile Malware in the Wild -," *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11*), ACM, New York, USA, 17, October, 2011.pp. 3-14

[27] K.W.Y Au, Y.F Zhou, Z. Huang, D. Lie, "Pscout: analyzing the android permission specification," *Proceedings of the 2012 ACM conference on Computer and communications*, ACM; New York, USA, 2012. pp. 217-228.

[28]W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," *Proceedings of 16th ACM Conference on Computer and Communications Security*, New York, USA, 2009, pp. 235-245.

[29] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," *Proceedings of* AsiaCCS'10, New York, USA, 2010, pp. 328-332.

[30]M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. "Semantically rich application-centric security in android," *Proceedings of the 25th Annual Computer Security Applications Conference*, ACSAC'09, Los Alamitos, CA, USA, 2009. IEEE Computer Society, pp. 340–349

[31] 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," *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, 2012/10/19, ACM, New York, USA, pp. 3-14,

[32]W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, Berkeley, CA, USA, 2010. pp. 1–6.

[33]P. F. Adam, A. Chaudhuri, and J. S. Foster, "ScanDroid: Automated security certification of android applications", In IEEE symposium of security and privacy, 2009

[34] 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," *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ACM, New York, USA. 2013, pp. 611–622.

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

[36] W. Xu, F. Zhang, and S. Zhu., "Permlyzer: Analyzing Permission Usage in Android Apps," *Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp.400-410.

[37] S. Feldman, D. Stadther, B. Wang, "Manilyzer: Automated Android Malware Detection through Manifest Analysis". *Proceedings of the 2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 767-772

[38] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P.G. Bringas y G, " Alvarez MAMA: Manifest Analysis for Malware Detection in Android," Cybernetics and Systems, v ?? n ??, ISSN (online): 1087-6553, ISSN (Print): 0196-9722 ID: 803889 DOI:10.1080/01969722.2013.803889

[39] Paolo Rovelli, Ýmir Vigfússon , "PMDS: Permission-based Malware Detection System" *Proceedings of the 10th International Conference on Information Systems Security (ICISS 2014)*. Lecture Notes in Computer Science (LNCS), Vol. 8880, pp 338-357, Springer, 2014.

[40] 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," *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS 2012)*, San Diego, CA, February 2012, pp. 317-326

[41] M. C. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day Android malware detection," *Proceedings of 10th International Conference on Mobile Systems*, Applications, and Services (MOBISYS), 2012, pp. 281–294.

[42] M. Z. Mas`ud, S. Sahib, M. F.l Abdollah, S. R. Selamat and R Yusof, "Android Malware Detection System Classification," Information Technology, vol: 6: pp. 325- 341. 2014.

[43] S. Neuner,V. Veen, M. Lindorfer, M. Huber, G. Merzdovnik, M. Mulazzani, E. Weippl, "Enter Sandbx: Android Sandbox Comparison," *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*, 2014, San Jose, CA, USA.