An Overview of Different Malware Analysis Techniques in Android

Aniruddh Bhilvare[1] Prof. Trupti Manik[2]
[1]M.E. Scholar [2]Assistant Professor
[1,2]Department of Information Technology
[1,2]L. D. College Of Engineering Ahmedabad, Gujarat, India

Abstract— This paper describes various malware analysis methodologies for android platform. Use of Smartphone has evolved a lot in recent times. Android is major operating system for mobile and tablet devices worldwide. Popularity of android has made if one of the most famous mobile device operating system. Along with this popularity Android has also attracted large number of malwares in the mobile world. The main aim of this paper is to Analyze and compare various malware detection methodologies available for detecting android malwares. It could then serve as the base to start with new research work by upcoming researchers.

Key words: Android, Mobile security, Malware analysis

I. INTRODUCTION

Android is an open source operating system for mobile devices based on Linux kernel develop+ped by Google. Unfortunately, the popularity of this platform also attracts malware developers. The malwares spread quickly in the Android market and cause financial loss or privacy leaking for mobile users. The openness of Android market allows malwares being published on the third-party Android markets without much inspection. This promotes the growth of Android malware.

Figure 1 shows architecture of Android OS, showing various components of operating system. Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown in the architecture diagram. At the bottom of the layers is Linux kernel. This provides basic system functionality like process management, memory management and device management like camera, keypad, display, sound, etc.

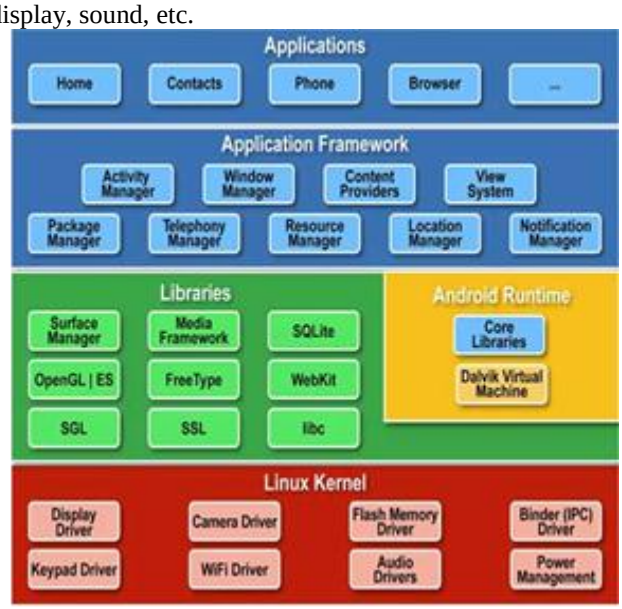

Fig. 1: Android architecture [1]

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, libc, SQLite database which is a useful for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc. Android runtime is the third section of the architecture and available on the second layer from the bottom. It provides a key component called Dalvik Virtual Machine (DVM) which is a kind of Java Virtual Machine specially designed and optimized for Android. DVM enables every Android application to run in its own process, with its own instance of the Dalvik VM. The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language. The Application Framework layer provides many higher-level services to applications in the form of Java classes. Thus, Application developers can use of these services in their applications. At the top of these stack, actual Android applications (Apps) which runs on mobile devices. Examples of such applications are Contacts, Browser, Calendar, etc.

Android has an extensive API and permission enforcement system. Every Android application has to define all the permissions required by the system APIs to perform certain tasks on device. When installing any application, the user is notified about the permissions required by the application. Users must have to grant the permissions requested by the application in order to run the apps.

II. OVERVIEW OF ANDROID PERMISSION CHECK SYSTEM

Android permission check system requires application developers declaring permissions used in their apps in *AndroidManifest.xml* file, in order to invoke any Android API successfully. Many literatures have discussed Android permission system [2], [3], [4]. An app can access Android API only if it has declared required permission its manifest file. For example, no app can access users' contacts on device unless it has declared READ_CONTACTS permission in its manifest file. Thus, these permission check system enforce some security for Android users. The declared permissions are useful and effective to reveal the potential risks of Apps being installed [6].

III. DIFFERENT ANDROID MALWARE ANALYSIS TECHNIQUES

Android malware analysis techniques can be categories in following:

- Static analysis
- Dynamic analysis
- Hybrid analysis

Static analysis checks for static behavior of apps like analysis of declared permissions in manifest file, source code of apps, etc. On the other hand, Dynamic analysis includes monitoring run time behavior of apps, collection and analysis of device traces, monitoring system calls, etc.

These both approaches are explained more details in next sections.

IV. STATIC MALWARE ANALYSIS

Static analysis mostly includes analysis of source code which is retrieved from apk of apps using various reverse engineering techniques.

- First, manifest file provides semantic information about the app. From declared permissions in manifest file it's possible to guess what kind of system APIs would have been used in source code. For example, if an app needs to listen to system-wide broadcast messages, the broadcast receivers can be statically contained in the manifest file or dynamically registered in the code. We can then check the following rule which means app is monitoring incoming SMS:
  android.provider.Telephony.SMS_RECEIVED
  Broadcast receivers are system level notifications which notify apps when defined rule is triggered. Also to use system APIs related to SMS functionalities RECEIVE_SMS permission must have declared in manifest file. Similar logic can be applied for analysis of all other declared permissions.
- Second, the app bytecode contains useful information, which can also be used in static analysis. Particularly, we can check what APIs are called, and their sequences in source code. By extending the previous example of app monitoring incoming SMS messages a call to the abortBroadcast function may potentially intercept received SMS messages. Furthermore, we can also utilize a data flow analysis algorithm to detect function parameters with static or fixed inputs. This method can be used to express malware behaviour such as sending SMS messages to premium numbers.
- Third, the structural layout of the app can also be analysed. As an Android app is essentially a compressed archive, we can decompress it to reveal its internal tree structure and then correspondingly analyse what packages are used by the app, what kind of class hierarchies they have, and where a specific resource is located.

Static analysis is applied on source code or bytecode for detecting malicious behavior of apps. However, if malicious app uses code obfuscation or Java reflection techniques then static analysis can't be useful.

Enck et al. [7] analyze the permissions requested by applications at install time and manually formulate rules to detect malicious applications. Similarly, Peng et al. [8] analyze permissions to infer probabilistic models for malicious applications. Felt et al. [9] go one step further by correlating requested permissions with statically observable API calls to detect over privileged applications. Unfortunately, these approaches are unable to detect malware which elevates its privileges by exploiting vulnerabilities [12], [10].

Static analysis can be deployed for malware detection in Android devices. But due to the limited resources of smartphones, most of the recent proposals for malware detection on Android devices are based on behavior analysis for anomaly detection.

V. DYNAMIC MALWARE ANALYSIS

Dynamic analysis is based on the behavior features of the applications which require running the applications in a controlled or isolated environment. We detect abnormal execution patterns through behavioral signatures or learning/mining techniques.

For example, Dixon et al. [16] and Liu et al. [11] monitor power consumption of a device for anomalies. Crowdroid [5] performs clustering algorithms on activity and behavior data of running apps to classify them as benign or malicious. DroidRanger [13] detects known malicious apps in Android Markets. It extracts behavioral signatures of known malware samples, and then detects new samples of known malware families using the behavioral signatures. TaintDroid [18] uses dynamic taint analysis to report potential privacy leaks at runtime. Rastogi et al. [15] proposes the AppsPlayground framework which performs dynamic analysis of Android apps for the purpose of detecting malicious activities and privacy leaks. Aurasium [19] repackages apps to add user-level sandbox and security policies so that the app's runtime behavior can be restricted. ParanoidAndroid [14] performs runtime analysis for malware detection on a remote server in the cloud. DroidScope [20] provides a virtualization environment for the purposes of dynamic analysis and information tracking of Android apps. AppFence [21] modifies Android OS to protect private data from being leaked by providing and imposing fine-grained privacy controls on existing apps. TISSA [22] proposes a privacy mode in Android platform which provides fine-grained control over user privacy.

VI. HYBRID MALWARE ANALYSIS

Resource limitations of smartphones have lead researchers to propose collaborative analysis techniques, where the analysis is made by a network of devices. This collaborative analysis can be categorized as hybrid malware analysis approach. In this, instead of going for either static or dynamic malware analysis combination of both schemes are used to detect malicious behavior of apps.

Antivirus companies have adapted their signature-based detection systems to smartphones, but considering the level of resources needed by antivirus techniques and the power and memory constraints of mobile devices, in-phone analysis is not a preferred solution to apply in smartphones. In this kind of approach, we collect data from devices like system calls or traces from android apps. Then those collected data is transferred to servers where actual analysis occurs. This way we can collect useful runtime data of apps and apply strong analysis algorithms on it for detecting malwares.

Schmidt et al. proposed a system in [23] an Android application sandbox. First they perform static analysis dissembling Android APK _les in order to detect Malware patterns. Then, dynamic analysis is carried out, executing and monitoring Android applications in a totally secure environment, also known as Sandbox. During dynamic analysis, all the events occurring in the device (opened _les, accessed _les, battery consumption, etc.) were monitored. The main drawback of their system is the use of an application that simulates user interaction (known as ADB Monkey), which will never be as real as a user.

Functions	Static Analysis	Dynamic Analysis
Analysis mode	Analyze app in offline mode.	Analyze app while in execution.
Malware analysis	– Various reverse engineering tools i.e. ApkTool and Dex2Jar are used to retrieve source code from apk of apps. – Here we check for declared permissions in *AndroidManifest.xml* file. – Analysis is done from suspicious patterns of certain malware family. – System APIs calls are analyzed to detect malicious behavior of apps.	– Analysis is based on the behavior features of the applications which require running the app. – Analysis is based on pattern checking of system calls and execution paths. – Dynamic privacy leaks and power consumptions are monitored to detect malicious behavior of apps.
Limitations	– In case of code obfuscation this method can't be effective. – Native code can't be analyzed in case of bytecode analysis. – Can't detect new malware families.	– As runtime behavior of apps is needs to be analyzed it can increases overhead on actual device.

Table 1: Difference between static and dynamic Android malware analysis

Method	Analysis approach	Classification Policy
Crowdroid [5]	Dynamic analysis	It performs clustering algorithms on activity and behavior data of running apps to classify them as benign or malicious.
DroidRanger [13]	Static analysis	It extracts behavioral signatures of known malware samples, and then detects new samples of known malware families using the behavioral signatures.
TaintDroid [18]	Dynamic analysis	It uses dynamic taint analysis to report potential privacy leaks at runtime.
Aurasium [19]	Dynamic analysis	It repackages apps to add user-level sandbox and security policies so that the app's runtime behavior can be restricted.
ParanoidAndroid [14]	Dynamic analysis	It performs runtime analysis for malware detection on a remote server in the cloud.
AppFence [21]	Dynamic analysis	It modifies Android OS to protect private data from being leaked by providing and imposing fine-grained privacy controls on existing apps.
SCanDroid [24]	Static analysis	It extracts security specifications from the app's manifest and checks whether data flows through the app are consistent with the stated specifications.
Stowaway [25]	Static analysis	It aims to identify over privileged Android apps by comparing the required and requested permissions based on mapping API calls used for permissions.
AdRisk [26]	Static analysis	It systematically studies large number of popular ad libraries used in Android apps to evaluate the potential risks.
DNADroid [27]	Static analysis	It detects cloned Android apps in the markets by comparing similarities of program dependency graphs.
RiskRanker [28]	Static analysis	It detect Android malware using, 1) a set of vulnerability specific-signatures and 2) control flow and intra-method data flow analysis looking for suspicious behavior signatures

Table 2: Comparison of various malware analysis solutions for Android apps

VII. Conclusion

This paper describes various Android malware analysis techniques and Comparison between them. The comparative study could make researchers better understand techniques for malware analysis in android and key difference between them.

References

[1] Architecture of Android operating system, http://www.tutorialspoint.com/android/android_architecture.htm
[2] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM conference on Computer and communications security, 2011, pp. 627–638.
[3] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, 2010, pp. 328–332.
[4] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou, "Analysis of android applications' permissions," in Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability Companion, 2012, pp. 45–46.

[5] BURGUERA, I., ZURUTUZA, U., AND NADJM-TEHRANI, S. Crowdroid: behavior-based malware detection system for Android. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM) (2011), ACM, pp. 15–26.

[6] 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, 2012, pp. 3:1–3:14.

[7] W. Enck, M. Ongtang, and P. D. McDaniel. On lightweight mobile phone application certi_cation. In Proc. of ACM Conference on Computer and Commu- nications Security (CCS), pages 235{245, 2009. [12] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri.A study of Android application security. In Proc. Of USENIX Security Symposium, 2011.

[8] H. Peng, C. S. Gates, B. P. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using probabilistic generative models for ranking risks of android apps. In Proc. of ACM Conference on Computer and Communications Security (CCS), pages 241{252, 2012}.

[9] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demysti_ed. In Proc. of ACM Conference on Computer and Communications Security (CCS), pages 627{638, 2011.

[10] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: statically vetting android apps for component hijacking vulnerabilities. In Proc. of ACM Conference on Computer and Communications Security (CCS), 2012.

[11] LIU, L., YAN, G., ZHANG, X., AND CHEN, S. VirusMeter: Preventing your cellphone from spies. In Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (2009), Springer, pp. 244–264.

[12] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In Proc. Of USENIX Security Symposium, 2011.

[13] ZHOU, Y., WANG, Z., ZHOU, W., AND JIANG, X. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In Proceedings of the 19th Network and Distributed System Security Symposium (NDSS) (2012).

[14] PORTOKALIDIS, G., HOMBURG, P., ANAGNOSTAKIS, K., AND BOS, H. Paranoid Android: versatile protection for smartphones. In Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC) (2010), ACM, pp. 347–356.

[15] RASTOGI, V., CHEN, Y., AND ENCK, W. AppsPlayground: Automatic large-scale dynamic analysis of Android applications. In Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY) (2013), ACM.

[16] DIXON, B., JIANG, Y., JAIANTILAL, A., AND MISHRA, S. Location based power analysis to detect malicious code in smartphones. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (2011), ACM, pp. 27–32.

[17] EGELE, M., KRUEGEL, C., KIRDA, E., AND VIGNA, G. PiOS: Detecting Privacy Leaks in iOS Applications. In Proceedings of the Network and Distributed System Security Symposium (NDSS) (2011), The Internet Society.

[18] ENCK, W., GILBERT, P., GON CHUN, B., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. TaintDroid: An informationflow tracking system for realtime privacy monitoring on smartphones. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (2010), USENIX Association, pp. 393–407.

[19] XU, R., SAÏDI, H., AND ANDERSON, R. Aurasium: practical policy enforcement for Android applications. In Proceedings of the 21st USENIX conference on Security symposium (2012), USENIX Association.

[20] YAN, L. K., AND YIN, H. DroidScope: seamlessly reconstructing the os and dalvik semantic views for dynamic Android malware analysis. In Proceedings of the 21st USENIX conference on Security symposium (2012), USENIX Association.

[21] HORNYACK, P., HAN, S., JUNG, J., SCHECHTER, S. E., AND WETHERALL, D. These aren't the droids you're looking for: retrofitting Android to protect data from imperious applications. In Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS) (2011), ACM, pp. 639–652.

[22] ZHOU, Y., ZHANG, X., JIANG, X., AND FREEH, V. W. Taming information-stealing smartphone applications (on Android). In Proceedings of the 4th International Conference on Trust and Trustworthy Computing (TRUST) (2011), Springer, pp. 93–107.

[23] Thomas Bl□asing, Aubrey-Derrick Schmidt, Leonid Batyuk, Seyit A. Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In 5th International Conference on Malicious and Unwanted Software (Malware 2010) (MALWARE'2010), Nancy, France, France, 2010.

[24] FUCHS, A. P., CHAUDHURI, A., AND FOSTER, J. S. SCanDroid: Automated security certification of Android applications, 2009. Technical report, University of Maryland.

[25] FELT, A. P., CHIN, E., HANNA, S., SONG, D., AND WAGNER, D. Android permissions demystified. In Proceedings of the 18th ACM Conference on Computer and Communications Security (2011), ACM, pp. 627–638.

[26] GRACE, M. C., ZHOU, W., JIANG, X., AND SADEGHI, A.R. Unsafe exposure analysis of mobile in-app advertisements. In Proceedings of

the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC) (2012), ACM, pp. 101–112.

[27] CRUSSELL, J., GIBLER, C., AND CHEN, H. Attack of the clones: Detecting cloned applications on Android markets. In Proceedings of the 17th European Symposium on Research in Computer Security (ESORICS) (2012), vol. 7459 of Lecture Notes in Computer Science, Springer, pp. 37–54.

[28] GRACE, M. C., ZHOU, Y., ZHANG, Q., ZOU, S., AND JIANG, X. RiskRanker: scalable and accurate zero-day Android malware detection. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys) (2012), ACM, pp. 281–294.

