The Evolution of Android Malware and Android Analysis Techniques (Summary)

DONGMO DIFFO Joel & BOIMADI Nicole

I- Introduction

1.1 Context

Nowadays, smartphones, tablets ands several mobile equipments are spread over the globe due to their personal and powerful attributes. As we going on, more and more sensitive information pass through devices and are targeted by malicious stakeholders. This is achieved via mobile malware, especially Android malware which become more sophisticated. The depiction above sets the context of the following article : "The evolution of android malware and android analysis techniques", Kim et al., 2017. This survey, by its authors, allows us to explore the relationship between malware and detection techniques to understand how newest malware and analysis techniques are linked and influenced each other.

1.2 Background

A) Evolution of mobile malware

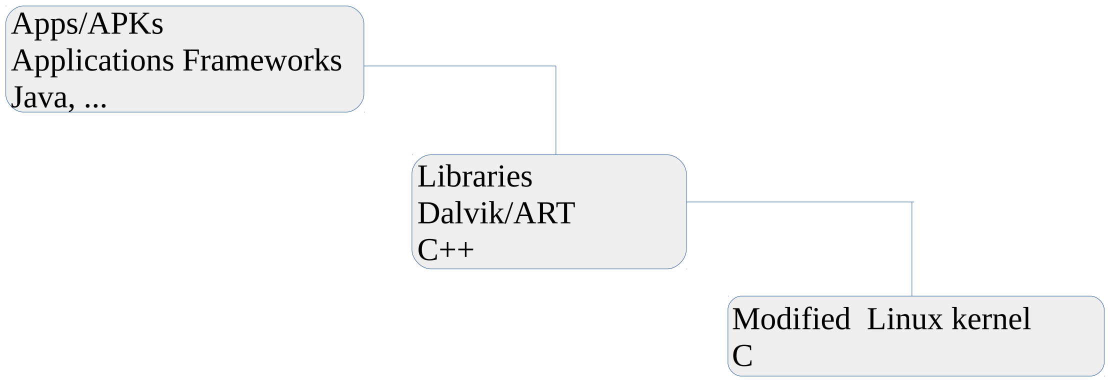
At the dawn of computing systems which essentially understand by few experts malware development was mostly a technical-skill test than a real threat. By progressing in time, this benign trait of malware vanishes as the great makable profit of all these private information grows up behind. According to the Q1 2014 F-Secure report, of all mobile malwares targeting Android devices we step up to 79% in 2012, to 96% in 2013 and 99% in the first quarter of 2014. One of the prime contributing factors to this immense malware growth is Android's open-source operating system and application markets. This includes the official Google Play, which has some vetting processes, as well as "unofficial" third-party markets across the world. At the end of 2014, McAfee also analyzed regional infections rates of devices running their security products. They found the infection rates in Africa and Asia were roughly 10%, while Europe and both Americas had rates between 6% and 8%.

Dealing with traits of android malware, with the extreme movements allowed by the mobile devices and the multiplicity of nearby installation sources, Android's are more exposed to malicious apps. Worms utilize their host's movements to propagate themselves across networks domains. Additionally, mobile devices are also vulnerable through connections to the outside : Wi-Fi, cellular networks (GPRS, LTE, Edge, …), bluetooth and web browsers. The profit-driven aspect of Android's malwares is illustrated by a large numbers of them in that background SMS are sent to premium numbers to generate revenues. Though attempts to mitigate this have been made in Android OS 4.3, released in 2012, more robust solutions such as AirBag [Xiang et al.2014] are still necessary but still insufficient. As an example, it was estimated that over one thousand devices were affected with one particular malicious version of the Angry Birds game. Once installed, the malware secretly sent premium SMS each time the game was started. Often, the target is more deeper as the malware use privilege escalation to exploit OS and Kernel vulnerabilities in aim to gain root access. In successful cases, it is granted access to the lower, higher-privileged, architectural layers and then leak data pertaining to device, owner or both. A more close function-based is spyware which monitors a target remotely via mobile devices. Similarly, malware can deny services of other apps (including anti-virus apps) by overusing resources (e.g., battery, bandwidth) and tampering with necessary files or processes.

From 2011 to 2015, there have been many malware families discovered but only a few worth mentioning. The majority of samples has been discovered between 2014 and 2015; The Android malware NotCompatible.C infected over four million devices to send spam emails, buy event tickets in bulk, and crack WordPress accounts. Conversely, malware such as Dendroid and Android.hehe are more difficult to analyze dynamically, as they are aware of emulated surroundings and have consistently evaded Google Play's vetting processes.

B) Android overview

Android OS was first released in 2008 based on a modified Linux kernel running all Java-written applications in isolation. Normally, this imply all apps are run separately in their own Dalvik Virtual Machine, but has been renewed with the Android 5.0 release which implemented the Android Runtime. To gain access to system, all apps should be granted permissions during installation by the Android Permission system. Android architecture is essentially composed as :



Depending on the way the functionalities are integrated and how they interact with the system Android differs from other mobile OS, principally :
-iOS;
-Windows OS;
-Palm;
-Blackberry;
-Symbian.

Finally, aware of all that sophisticated mobile malware, particularly Android malware, acquire or utilize such data without user consent. It is therefore essential to devise effective techniques to analyze and detect these threats. Provided by Kimberly Tam et al., this article presents a comprehensive survey on leading Android malware analysis and detection techniques, and their effectiveness against evolving malware. This article categorizes systems by methodology and date to evaluate progression and weaknesses. This article also discusses evaluations of industry solutions, malware statistics, and malware evasion techniques and concludes by supporting future research paths.

II- Related works

Table : Comparative classification of recent surveys

Surveys	**Background**	**Threat**	**Static Analysis**	**Dynamic Analysis**	**Malware tactics**
Purpose article	Traditional + Android	Discussion + small study	Comprehensive coverage	Comprehensive coverage	Obfuscation + evasiveness
[Xu et al. 2016]	Android ecosystem	Privilege escalation	Dataflow and taint analysis	Dataflow + mentions studies	Obfuscation, moderate detail
[Rasthofer et al. 2016]	✗	Detection's counter	Runtime value	Dataflow	Obfuscation, explicit detail
[Sulfatrio 2015]	Android	Complete taxonomy	Mention studies, detail	✗	✗
[Faruki et al. 2015]	Android	malware types and actions	Moderate coverage	Moderate coverage	Obfuscation
[Huang et al. 2015]	Android	privacy leaks	Mention	Dataflow	✗
[Tchakounte et al. 2015]	Android ecosystem	Privilige escalation	Permissions, moderate coverage	Moderate coverage	Moderate detail
[Li et al. 2015]	Moderate coverage	Privacy leaks	ICC taint analysis	✗	✗
[Suarez et al. 2014]	Smartphones	Malware attacks	Moderate coverage	Mention studies, detail	Obfuscation, less detail
[Arp et al. 2014]	✗	Android malware	Tait analysis, vectors, learned-based	✗	✗
[Gianazza et al. 2014]	Analysis tech.	Android malware	✗	Stimulations traces	✗
[Mahmood et al. 2014]	Android	Testing app	✗	Code segments	✗
[Petsas et al. 2014]	Taint analysis, android	Malware tactics	Mention approach	Sensor information	Obfuscation, more detail
[Tchakounte 2014]	Android Permission system	Permission-based attacks	Tool review, machine learning	✗	✗
[Teufl et al. 2014]	Mobile malware, android	Mobile attacks	Mention studies	Mention studies, moderate coverage	✗

[Polla et al. 2013]	Mobile tech.	Attacks + evolution	Mention studies, detail	✗	✗
[Hyo-Sick et al. 2013]	Android	Malware	machine learning	✗	✗
[Jin et al. 2013]	Smartphones	Mobile malware	✗	SDN-based	✗
[Maggi et al. 2013]	Android	Malware evasiveness	Moderate coverage	Mention studies	Obfuscation, repackage
[Zhou et al. 2013]	Third-party markets	Repackaged apps	Moderate coverage	Less detail	Repackaging, re-routing
[Amamra et al. 2012]	Taint analysis	Classified taxonomy	✗	✗	✗
[Branco et al. 2012]	Analysis approaches	Mobile malware	Signature-based	Behavior-based	✗
[Zhou et Jiang 2012]	Android malware	Characterize malware	✗	✗	Obfuscation mentioned
[Enck 2011]	Traditional+ android	Malware attacks	Mainly dynamic taint analysis	Details a few methods	✗
[Vidas et al. 2011]	Android	Attacks vectors	✗	✗	✗
[Felt et al. 2011]	Android perm.	Permission abuse	✗	Details of used (one method	Challenges for used method
[Burguera et al. 2011]	Android platform	Malware analysis	✗	Crowdsourcing	✗
[Becher et al. 2011]	Smartphones	mobile networks	Dynamic taint analysis	Function call analysis	✗

III- Methodology employed

This part is mainly composed of two sections. First the study of android malware analysis frameworks by methods, year and outcome; to identify and expose strengths, weaknesses, performance and uses of techniques in similar studies. Second, an overview of diverse android malware tactics employed to evade/escape analysis, establishment of classification and description of several advanced malware evasion techniques.

3.1 Taxonomy of mobile malware analysis

Static analysis
By proceeding statically, we examine the program without executing any code. However, all static methods are vulnerable to obfuscations that limit or remove access to the code. Due to the fact android app source code is rarely available, so many frameworks analyze the app bytecode inside the package APK. The main Android APK components for static are : the AndroidManifest.xml, *classes.dex*, permission, intents, hardware component, dex files.

Dynamic analysis
In-the-box (In Guest) Analysis : the examination and gathering data occurs on the same privilege level as the malware. Therefore, despite its great advantage that memory structures and high OS-level data are easily accessible, the downfall of in-guest analysis is that due to the "close proximity", the application to be analyzed can attack or bypass the framework.

Out-of-the-box Analysis : This approach is essentially base on Virtual Machine and emulation in aim to guaranty increased security via simulation. Unfortunately, malware can, and has, countered emulation by detecting non-realistic environments; and then stop its execution or act hiding the malicious features like a benign app.

Virtualization. It is a partial emulation lighter than a full one but, if well-implemented, still provide robust security.
Hybrid Analysis can be a solution in order to improve robustness, this approach greatly contribute to cover the lacks of either static or dynamic analysis.

3.2 Malware analysis approaches

Besides of being static or dynamic, the different techniques can be identify and are unique each other. According to the technique implemented, we can differentiate : network traffic, application programming Interfaces (API), systems calls, dependency graphs, features, function call monitoring, Information Flow, Inter-Process Communications Analysis, Hardware analysis, Android Application Metadata.

To improve the accuracy of the research, we may select the appropriate feature from all offered by the growing sophisticated Android's apps. There are principally two ways to do such, *selection reasoning*, which tend to perform the analysis based on a set of features; *feature ranking algorithms* by identifying the ideal feature.

3.3 Evolution of Malware Tactics

In this article, static obfuscation techniques are split up in three categories : trivial transformations, transformations that hinder static analysis and transformations that can prevent static analysis.

Trivial transformations deter mainly signature-based analysis, proper to the Android framework, unzipping and repackaging APK files is a trivial form of obfuscation without any modifcation of the manifest. APK dex files may also be decompiled, and reassembled.

Even some techniques are resistant to obfuscation, there is still some vulnerabilities specific to each method. Notably, we denote here data obfuscation and control flow obfuscation. Data obfuscation proceeds by renaming app methods, classes and field identifiers or even the Manifest's package. In Android, malicious native code can be stored within the APK. Additionally, control flow method deters call-graph analysis with call indirections. For example, as Java bytecode possesses the goto instruction while normally Java does not. Bytecode can then be scrambled with goto instructions inserted to preserve runtime execution. Function inlining, the breaking of functions into multiple smaller functions, can be combined with call indirections to generate stronger obfuscation.

Dealing with techniques that prevent static analysis, they are, most of time, based on the encryption of relevant parts of the Android's APK. Reflection as well as cryptography are useful tools for obfuscation. Also, *dynamically loaded code* cannot be analyzed statically and difficult dynamically. It loads a library into memory at runtime.

Apart of those main categories, we note anti-analysis methods that detect and evade emulated environment by identifying missing hardware and phone identifiers. When detecting any traces of emulation, malwares then hide their malicious behaviors or for other families, counter the anti-virus software. These are methods that complicate dynamic analysis, as well as data obfuscation (like encryption), misleading information flows, mimicry and function indirections.

IV- Results found and Perspectives

Based on the study the authors conducted, the samples were made of the combination of the DREBIN project and the Malware Genome project. The analysis lays on the Androgauard tool, hence the ,main script could compile data on how many malware in our dataset used techniques like native code. This is to determine how, through time, Android malware evolved to avoid analysis.

Android Malware Obfuscation: after analyzing more than 9300 Android malware samples, the team the INTERNET and ACCESS_NETWORK_STATE are the top permission ranked. Despite the large static analysis methods that have been created, trends show a increase in the usage of dynamically loaded code and Java reflection. And though dynamic analysis is more robust against use of dynamic evasions, its effectiveness is often reduced by its limited code coverage.

Permission Usage and Malware Threat: within the dataset, apart from the INTERNET permission, the following is READ_PHONE_ESTATE (such as IMEI, IMSI, phone number) and then the malware to gather these data. Also, the WRITE_SETTINGS permission jump of 8.5% (2010) to 20.38% (2014) on one hand, and the SYSTEM_ALERT_WINDOW at 0.23% in 2010 rocketed up to 24.8% in 2014. Once granted, these permissions can be very dangerous as allowing malwares to perform malicious exploits. Several new witnessed permissions such as MOUNT_FORMAT_FILESYSTEMS, USE_CREDENTIALS and AUTHENTICATE_ACCOUNTS are dangerous permission as they could greatly aid in privilege escalation. That is, in part, the cause of the percentage of malware in the dataset increased from 69% in 2010 to 79% by 2014.

Impact and Motivation. By gathering and analyzing diverse Android malware techniques qnd Android malware analysis frameworks, this survey has identified several risks that should motivate continuous research efforts in certain directions.

-<u>Malware Growth and Infections</u>: when considering that the majority of new malware are undetected by antivirus products, it is highly plausible that actual infection rates are higher than reported. To reduce malware infections, malware markets need to be able to both accurately vet submitted applications and remove available malware as soon as they have been identified or detected by themselves or by an external party.

-Weaknesses in Analysis Frameworks: As discussed through the article, many frameworks today are unable to analyze dynamically loaded code and are vulnerable to at least one kind of obfuscation. Class loaders, package content, runtime Java class are the common methods to run root exploits.

-Weaknesses in Anti-Virus Products: As shown early in the article, signature-based AV products can be broken by the simplest transformations. Although trivial transformations are someway weak in front a dynamic method, some heavily obfuscated malware, against static analysis, can also bypass dynamic analysis.

-Lack of Representative Datasets: even since the first Android malware discovery, researchers lacked a solid, standard dataset to work with. As the nature of Android malware constantly evolve, it is essential to update the dataset with newer samples to keep the detection's systems efficient.

-IoT: another interesting point of discussion is the Internet of Things, the concept of all elements our lifehood are interconnected (gate, door, lights, TVs, fridge), principally based on a simple, open-source and reasonably sized OS, such as Android. If the IoT were to adopt smaller, altered versions of the Android OS, then it would give researchers an incentive to create portable analysis and detection tools so they may be usable across all Android OS versions no matter what device it powers.

Mobile Security Effectiveness. To evaluate the present (we are here in 2016') status of Android malware analysis and detection frameworks the article we discuss here provides some recap in its section 5 for further details.

-Analyzed Datasets: many studies only use one app source or select apps that best suit their research; however, ideally malware samples should be chosen from several families to provide a more diverse set of behaviors including evasion techniques, with which to test the system.

-Scalability, Accuracy, and Portability: it is vital for a detection system to be scalable as the body of malware grows and diversifies. While most systems scale well enough, some do trade scalability for accuracy and visa versa. The authors underline here that several systems were able to detect, but not to analyze, samples with traits such as native code, Java reflection, VM-awareness and obfuscation.

-Significant Changes in Android: as mentioned above, one of the most relevant changes to Android has been the switch from the Dalvik runtime to the ART runtime. Ideally, solutions should be agnostic to those parts changed through the released versions. However, many static solutions rely on the Dalvik dex file, as opposed to the new odex files, and many dynamic solutions either modify or are very in tune with specific aspects of Dalvik runtime internals.

Future Research Directions. In summary, Android malware analysis is most tends into the right direction. The following areas for future research are suggested.

-Hybrid Analysis and Multi-leveled Analysis: as static analysis are beginning to harden against trivial obfuscations and most malware are already using higher levels of obfuscation; in case of obfuscation, dynamic analysis can be used in conjunction. Hybrid solutions could therefore combine static and dynamic analysis in ways that their added strengths attenuate each other's weakness such as the Harvester tool. Another way is to develop multi-level system in such an aim that it would be harder for a malware to hide actions if multiple layers of the Android architecture are being monitored, despite of additional overhead, decrease transparency, increased chances of code bugs and less portability.

-Code Coverage: to achieve robust malware analyses, code coverage is essential. Dynamically, user interactions are difficult to automate, thus the set of paths per execution are limited. While hybrid solutions and smarter stimulations (IntelliDroid, a static and dynamic API-based input generator) would greatly increase code coverage, different approaches should be further researched based on malware trends. Another topic that may be beneficial is identying and understanding subsets of malware behaviors through path restrictions to see which behavior equates to which permission and/or trigger.

-Hybrid Devices and Virtualization: apart adding smart stimuli, modifying emulators for increase tranparency or using emulators with access to real physical hardware (sensors, accelerometers) to fool VM-aware malware may prove useful and interesting. However achieving a perfect emulator is idealogical, malware such as DenDroid and Android.hehe do not just detect their emulated environment but also hide their malicious behaviors or tamper with the environment. One solution to this problem would be to use real devices in all dynamic experiments. Here the authors especially propose combining real devices and emulators as a new hybrid solution; this would ideally reduce the cost ans speed of experiments while revealing more malicious behaviors. As an alternative to virtualization, it would also be interesting to see if splitting the kernel, where untrusted system calls are directed to the hardened kernel code, can be applied. Finally, we look toward new technology, such as the new ARM with full virtualization support, and more explorations into ART and its new challenges.

V- Most important publications for this work

Here are recent additional surveys related to the Android malware universe and analysis techniques.

[1] Hossain Shahriar and Victor Clincy, Detection of repackaged Android Malware, February 2015.

[2] Pham Than Giang et al., Permission Analysis for Android Malware Detection, *in* The Proceedings of the 7th VAST - AIST Workshop "RESEARCH COLLABORATION: REVIEW AND PERSPECTIVE", At Hanoi, Vietnam, November 2015.

[3] Franklin Tchakounte et al., A Malware Detection Systen For Android, September 2015.

[4] Sapna Malik and Kiran Khatter, System Call Analysis of Android Malware Families, June 2016.

[5]Murat Yesilyurt et al., Security Threats on Mobile Devices and their Effects: Estimations for the future, 2016.

[6] Shahid Alam et al., DroidNative: Semantic-Based Detection of Android Native Code Malware, February 2016.

[7] Nishant Painter and Bintu Kadhiwala, Comparative Analysis of Android Malware Detection Techniques, January 2017.

[8] Ankita Kapratwar et al., Static and Dynamic Analysis of Android Malware, *in* 1st International Workshop on FORmal methods for Security Engineering, January 2017.