Digital Investigation xxx (2018) 1–9

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

An in-depth analysis of Android malware using hybrid techniques

Abdullah Talha Kabakus [a, *], Ibrahim Alper Dogru [b]

[a] *Duzce University, Faculty of Engineering, Department of Computer Engineering, 81620, Duzce, Turkey*
[b] *Gazi University, Faculty of Technology, Department of Computer Engineering, 06560, Ankara, Turkey*

A R T I C L E I N F O

Article history:
Received 29 September 2017
Received in revised form
26 December 2017
Accepted 10 January 2018
Available online xxx

Keywords:
Android malware
Android
Play store
Mobile security
Malware analysis

A B S T R A C T

Android malware is widespread despite the effort provided by Google in order to prevent it from the official application market, *Play Store*. Two techniques namely static and dynamic analysis are commonly used to detect malicious applications in Android ecosystem. Both of these techniques have their own advantages and disadvantages. In this paper, we propose a novel hybrid Android malware analysis approach namely *mad4a* which uses the advantages of both static and dynamic analysis techniques. The aim of this study is revealing some unknown characteristics of Android malware through the used various analysis techniques. As the result of static and dynamic analysis on the widely used Android application datasets, digital investigators are informed about some underestimated characteristics of Android malware.

Introduction

Smartphones have changed the life of people dramatically in the last decade thanks to the provided functionalities and mobility. Android leads the mobile operating system market by being used on over 2 billion monthly active devices (Burke, 2017; Popper, 2017). According to a recent report by *IDC*[1], Android dominates the global smartphone market with being used on 85% of smartphones in all around the world (IDC Smartphone OS Market Share, 2017). It is expected that Android's global market share is expected to rise to 90% in 2017 (Bosnjak, 2017). As a result of this popularity, the official application market, *Play Store*, is used to install 82 billion applications in 2016 (Burke, 2017). It is reported that *Play Store* is growing at three times the rate of *Apple*'s *App Store* which is the official application market of *iOS* and the biggest official mobile application market after *Play Store* (Lookout, 2011). As a result of this popularity, *Play Store* attracts the attention of malware developers (Delac et al., 2011; Portokalidis et al., 2010; Wu et al., 2012; Zhou et al., 2012). Android malware has grown by 580% between September 2011 and September 2012 (Protalinski, 2012). According to a recent report by *Check Point*[2], the Android malware app "*Judy*" may have reached as many as 36.5 million users (The Judy Malware Possibly the largest malware campaign found on Google Play, 2017). *McAfee Labs* report that there are around 2.5 million new Android malware samples exposed yearly (McAfee Labs Threats Predictions Report, 2016). Also, they report that total mobile malware grew 79% in the past four quarters to 16.7 million samples (McAfee Labs Threats Report June 2017, 2017). Despite that these reports demonstrate how serious the threat is, the lack of security awareness of Android digital investigators is reported by many researches (Enck et al., 2009; Kelley et al., 2012; King et al., 2011; Mylonas et al., 2013). According to a recent report, while only 17% of participants are interested in permissions while installing the applications, 42% of participants are even unaware of the permissions (Felt et al., 2012). Google uses *Bouncer* which is a service supposed to detect malicious applications which are available on *Play Store* by scanning every available application using dynamic analysis (Alzaylaee et al., 2017; Lockheimer, 2012). Alongside to the *Bouncer*, Google has announced *Google Play Protect* during the event *Google I/O 2017* (Android – Google Play Protect, 2017; Cunningham, 2017). *Google Play Protect* is an always-on service which is bundled with the *Play Store* app. *Google Play Protect* scans the applications automatically even after the installation to ensure the applications remain safe in terms of security. According to the official website of *Google Play Protect*, it is reported that 50 billion applications are scanned by *Google Play Protect* daily (Android – Google Play Protect, 2017). An advantage of *Google Play Protect* over *Bouncer* is that *Google Play Protect* is able to scan applications which are not

* Corresponding author.
E-mail address: talhakabakus@gmail.com (A.T. Kabakus).
[1] http://idc.com.
[2] https://checkpoint.com.

https://doi.org/10.1016/j.diin.2018.01.001
1742-2876/© 2018 Elsevier Ltd. All rights reserved.

installed from *Play Store*. To the best of our knowledge, this paper is the first academic paper which introduces the *Google Play Protect*.

Android malware detection systems are generally categorized into two: (1) Static analysis, and (2) dynamic analysis. Both of them have own advantages and disadvantages as it is discussed in Section 3. To combine the advantages of each analysis technique, we propose a hybrid Android malware analysis framework namely *mad4a* which stands for “Malicious Application Detector for Android”. The main objective of this study is revealing the characteristics of Android applications through the proposed framework named *mad4a* which combines static and dynamic analyzing techniques in order to detect malware in Android. We investigate a large variety of Android applications in order to make a conclusion about the characterization and behavior of Android applications. The rest of the paper is structured as follows: Section 2 presents the related work. Section 3 discusses the proposed framework in detail. Section 4 discusses the findings and the result. Finally, Section 5 concludes the paper with future directions.

Related work

The related work can be classified through the technique it uses as follows: (1) Static analysis techniques, and (2) dynamic analysis techniques.

Static analysis

Feizollah et al. (Feizollah et al., 2017). propose an analysis of the effectiveness of intents for identifying malicious applications. They report that intents are a more valuable feature than permissions in terms of detecting Android malware. According to their evaluation, on an average, while an infected application declares 1.18 intent-filters, a benign application declares 1.61 intent-filters. Their proposed approach performs analysis on the smartphones. Due to the lack of both computation and storage resources, and power, *mad4a* is intentionally designed to perform analysis on a remote server. *RiskRanker* (Grace et al., 2012) is a scalable framework which utilizes various static analysis techniques such as the evaluation of program control flow graph and bytecode signatures. *Stowaway* (Felt et al., 2011a) detects the overprivilege by determining the set of API (Application Programming Interface) calls that an application uses which are mapped to the related permissions. They have evaluated *Stowaway* using a set of 940 applications and have found that about one-third of these applications are overprivileged. *Dendroid* (Suarez-Tangil et al., 2014) uses a text mining approach in order to analyze the code chunks in Android malware families. A high-level representation of the Control Flow Graph (CFG) is extracted using the detected code chunks instead of focusing on the specific sequence of instruction in the code chunks. The samples are classified into Android malware families by adopting the standard Vector Space Model and measuring the similarity between malware samples. *Peng* et al. (Peng et al., 2012). propose a static analysis approach solely based on permissions. They discuss the importance of effectively communicating the risk of an application to digital investigators. Also, they propose to use probabilistic generative model for risk scoring which they introduce. Schmidt (Schmidt, 2011) proposes a static analysis approach which uses the amount of free RAM (Random Access Memory), user inactivity in the last 10 s, the number of running processes, the percentage of CPU (Central Processing Unit) usage, and the number of SMS (Short Message Service) messages sent. *Nauman* et al. (Nauman et al., 2010). propose *Apex*, a policy enforcement framework for Android that allows a user to selectively grant permissions to applications as well as impose constraints on the usage of resources. *Apex* enables dynamic permission revocation which is also enabled with the release of Android 6.0 (API Level 23). *Kirin* (Enck et al., 2009) is a static analysis tool which evaluates application's permissions to perform lightweight certification to mitigate malware at installation time. *APK Auditor* (Kabakus et al., 2015) is a permission-based Android malware detection system which consists of three components namely (1) a central server, (2) a signature database, and (3) the Android client to interact with the server to scan applications for threats. *APK Auditor* calculates a malware score based on the requested permissions and then calculates the malware threshold limit dynamically using logistic regression. Finally, *APK Auditor* classifies the application as malicious if the calculated application malware score exceeds the malware threshold limit.

Dynamic analysis

Mahmood et al. (Mahmood et al., 2012). present an approach that utilizes *Robitium* test automation in order to test Android applications automatically in the cloud. The biggest limitation of using *Robotium* framework is that it requires the tested application to be signed in debug mode which is rarely used with the production-ready applications (Bierma et al., 2014). Even though applications which are not signed in debug mode can be resigned, this approach prevents these resigned applications to be distributed in Play Store. Unlike that work, *mad4a* does not have a limitation like that. *AppsPlayground* (Rastogi et al., 2013) is an automated dynamic analysis tool for Android applications. *AppsPlayground* uses permissions, and API calls. *MADAM (a Multi-level Anomaly Detector for Android Malware)* (Dini et al., 2012) is a dynamic analysis tool which concurrently monitors Android at both kernel and user levels in order to detect malware infections. *MADAM* exploits machine learning techniques to distinguish between benign and malicious behaviors. The features *MADAM* uses for the kernel-level analysis are system calls, running processes, memory and CPU usage. The user-level features *MADAM* uses are user-state, keystrokes, called numbers, sent or received SMS, and Bluetooth/Wi-Fi analysis. While monitoring and analysis processes of *MADAM* are performed on the local device, *mad4a* is specifically designed to perform the analysis on a remote server considering the limited resources (e.g., memory, CPU, disk space, battery) of smartphones. *Crowdroid* (Burguera et al., 2011) is a behavior-based dynamic analysis tool which monitors and analyzes system calls per application. Some dynamic analysis approaches (Buennemeyer et al., 2008; Jacoby and Davis, 2004; Kim et al., 2008) use the power consumption as the main malware detection feature for their analysis. Those approaches may be useful for the attacks which target power consumption but it is not sufficient since there are lots of different malware types (Alzaylaee et al., 2017). *mad4a* uses both static and dynamic features in order to cover as many malware types as possible. *TaintDroid* (Enck et al., 2010) is a system-wide information flow tracking tool that can simultaneously track multiple sources of sensitive data such as variables, methods, file, and messages throughout the program execution. According to their evaluation of 30 random and popular applications which are selected from *Play Store*, 15 applications have reported the location of users' to a remote advertising server. *Paranoid Android* (Portokalidis et al., 2010) transfers the recorded execution trace which is recorded on the smartphone to the cloud server over an encrypted channel. The cloud server replays the execution trace within the emulator. *Paranoid Android* uses a network proxy to connect to the Internet in order to intercept inbound traffic. Instead of using a proxy, *mad4a* accesses the network log file related to the simulated application which is located on the device.