MHEye: A Hybrid Android Security Assessment Tool for Ordinary Users

Mohammadreza Hazhirpasand
Software Composition Group
University of Bern, Switzerland
`http://scg.unibe.ch`

Abstract

Android users are often overwhelmed by security issues arising in the apps they use. Although malware analysis tools exist, they are challenging to adopt for average users. To avoid burdening mobile devices with complex and computationally expensive malware tools, we propose a hybrid approach that combines static and dynamic analyses, and distributes the analysis mainly on another device than the phone. We first review the Android architecture and several of the key security issues it faces, and we summarize existing approaches to malware detection. We conclude with a research plan to explore and develop a more user-friendly approach to malware detection for ordinary users.

1 Introduction

The constant growth of smartphone applications and sales has had a great impact on our daily lives, and this fact has encouraged attackers to develop malicious software for attacking mobile devices. Amongst all smartphone OSs, the Android OS is the most widely-used platform and its source code is publicly available. Furthermore, a great number of interesting Android devices run obsolete versions of the OS. These factors make Android smartphones attractive targets for malware authors. According to the F-Secure report, 79% of malware designed to target smartphones in 2013 chose Android OS as their target, and by employing advanced anti-detection techniques in malware, the effectiveness of anti-malware programs decreased from 95% to 40% [MAC$^+$17, CT05]. In 2015, Symantec reported a 230% growth of malicious apps on Android platform that utilized techniques to bypass existing anti-malware signature-based approaches [sym16]. The most frequently highlighted techniques used by malware in the Symantec report include obfuscation of application code, and detection of the sandbox and virtualized environments.

To battle these types of malware, new research paths have been explored to enhance malware analysis techniques in mobile environments [MCW$^+$17, ZZC17, BK17]. The malware family classification method uses samples and clues to classify malware into different groups or families. Malware from the same family should have similar intentions, exploit the same weaknesses and present the same behaviour, and this helps save time for malware analysis.

In practice, malware analysts use both static and dynamic analysis techniques to identify and extract clues from malware. Since static analysis methods do not depend on running the application, they tend to be less useful for analyzing obfuscated code. On the other hand, dynamic analysis methods overcome this shortcoming by analyzing applications while they are running.

We outline the previous work that has widely influenced detection of malware and malicious applications, and then briefly explore possibilities for future work combining and applying both static and dynamic analysis while an ordinary user uses his phone. We propose the following three research questions for future research:

- **RQ**$_1$: What are the possible ways for an Android malware developer to achieve their goals?
- **RQ**$_2$: How can these malicious files be identified by current tools and frameworks, and how efficient they are in terms of resource usage, time and knowledge required?
- **RQ**$_3$:What improvements can help existing frameworks in the automatic detection of malicious applications without the need of a security expert?

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2018 (sattose.org).
04-06 July 2018, Athens, Greece.

In section 2 we explain the current state of Android structure and types of security issues. Then, in section 3 we explain the existing approaches for detection of malware as well as introducing well-known frameworks and tools which are typically used in reverse engineering of android applications. Finally, we represent a brief perspective in section 4.

2 Android Architecture and Security Issues

This section briefly assesses the Android structure and security issues related to each part of the OS. According to our first research question, by gaining a good understanding of how an application executes, and what are the existing threats, a malware developer can write some destructive code and target this platform.

In the Android OS, applications are written in Java while shared libraries are created in C/C++. Any Android application must be compiled to Dalvik bytecode which runs under the Dalvik Virtual Machine (DVM). Each application is compressed into an APK file which contains several files including images, strings and the source code of the application. Also, an interesting file called AndroidManifest.XML is located in an APK archive file, and provides fruitful information regarding permissions, activities, services and broadcast receivers. The Dalvik bytecode source code is stored in a file called classes.dex and serves as an entry point of the application, to be be executed on DVM. Many reverse engineers and experts try to decompile this file to have access to the source code of the application.

In Android, Discretionary Access Control (DAC) distinguishes each process by a Unique ID within an isolated space [PFB+15]. Each application must be digitally signed in order to create trust relationships between applications and to guarantee that the application came from the author of an application. In case of sharing or disclosing a certificate, application A could use the certificate to sign itself as well as application B. As a result, both applications have access to their private files, codes, and manifest permissions.

Generally, an Android application consists of various elements which are briefly explained below: *Activity*: whatever is visible to a user in an application is contained in an activity, and an application could have many activities based on the developer's need. *Service*: as its name implies, background and long-running tasks in an application should be run as a service. *Broadcast Receiver*: this is where an application can interact with system-generated events or application-defined events. *Content Provider*: when one application wants to request a data from another application, the data should reside in a Content Provider.

These elements in an application are interesting for malware authors to perform their malicious code on the Android platform. They are the main focus of existing assessment tools to discover any suspicious activity.

In addition, there are various kinds of threats against these elements and the Android kernel that are reported and used by different applications in order to perform malicious activities on the user's phone.

Privacy Escalation: gaining root access is a popular way for malware to access protected resources on Linux machines and also Android phones. Researchers and attackers usually analyse the kernel and shared libraries to discover vulnerabilities to escalate their privilege.

Private Information Leakage: it is very common to gain access to private information of the user as everyone's phone is a place where messages, phone calls and many other types of data are stored.

Malicious Application: applications that pretend to be legitimate can earn money in the background and subscribe to SMS channels without the user's approval.

Colluding: when several applications are signed with the same certificate, they have the same UID and are capable of sharing permissions and code. Malware writers have great incentives to create colluding malware, for instance two applications by the same developer where the first one obtains internet access and the second one obtains location permission. Since the first application has access to the internet, it is feasible for this one to send the location of the user as well through the internet.

Denial of Service: an application can drain resources of a phone and perform a DoS attack against either a target machine or another user's phone.

Dangerous Permissions: Permissions are classified in Android Marshmallow 6.0 into two normal and dangerous categories. Once an application requires one of the dangerous permissions, the application has access to critical resources of user's phone. Android 6.0 and higher asks users for granting any dangerous permissions; however, there are many old devices that are not capable of getting newer versions of Android. Moreover, many users do not read carefully what permissions an application asks them.

Failure of two-factor authentication: Many applications need to have access to read SMSs, and this creates various security threats. One of the major threats affects the reliability of two-factor authentication systems that utilize an SMS channel as their second factor of authentication. Once a user has received a One Time Password (OTP) password, the malicious application is able to send it immediately to the hacker's server.

Although Android has experienced a lot of improvements such as preventing stack buffer overflow, integer overflow and added features like Address Space Layout Randomization (ASLR), there are still different approaches for malware writers to bypass or achieve to their goal.

3 Malware Detection and Existing Frameworks

In Android, there are many well-known and widely-spread worms, ransomware, trojans and backdoors that have convinced researchers and security experts to create different tools in order to dissect these malicious applications. Researchers use two broad categories of methods, *i.e.,* static and dynamic analysis, to analyse applications. In static analysis, there are different techniques like signature-based, component-based, permission-based, and Dalvik Bytecode analysis, and converting Dalvik code into Java Bytecode, which help researchers to analyse an application from various perspectives. In dynamic analysis, methods such as profile-based anomaly, behaviour-based and virtual machine introspection are the most popular ways to dynamically examine an application.

The security of malicious applications can be assessed in different places. A lightweight security and risk assessment could be accomplished on-device. On-device malware detection has some limitations. First anti-malware applications, just as other applications, run under normal privilege, hence this prevents them from scanning private files or memory of other applications. While system resources such as battery usage are a major concern for every user, background services by anti-malware applications may consume a lot of system resources by performing various tasks. Moreover, background services can be terminated by applications that acquire special permissions. Android phones are not shipped with root access, and without having access to the root account, anti-malware applications cannot hook into system calls or monitor network activities.

The other method is to distribute analysis work between the phone and another device. In this approach, high-cost computational work could be offloaded from the phone while basic detection can be performed on the device. In this method, there are some limitations such as continuous internet usage, bandwidth issues and resource usage.

With the last method to obtain a deep understanding of a malicious application sample, the whole process must be done somewhere else than a smartphone. The main reasons for choosing to perform analysis off-device could be (i) the need for human interaction in the static analysis in a more precise manner, (ii) developing some automated analysis modules, and (iii) the necessity of greater computational power and memory.

We are now going to review several tools and frameworks that are widely used by security researchers. The first one is *Apktool*, which can decode an APK file and disassemble the binary resources [PFB+15]. *Dex2jar* is a popular disassembler that converts a Dex file to a Jar file for any additional assessment or manipulation [BHM+15]. *Dare* is also another useful tool which is claimed to be 40% more accurate than Dex2jar [BGM+16]. Dare is capable of converting a Dex file to a traditional .CLASS files for any additional inspection. Among many disassemblers, *Dedexer* translates a Dex file into assembly-like format syntax and creates an easy way for a human to track the files and source-code [CFGW11]. *Androguard* is a static analysis tool that uses the off-device approach to generate control flow graphs of an application and some features that are accessible through a Python API [Nez17]. This tool also explores similarities and differences between different applications. *Andromaly* runs half on-device and half off-device and utilizes machine learning to monitor real-time usage of CPU, memory and network activity [SKE+12]. Andromaly has a graphical user interface to configure what parameters and options should participate in the detection phase. *APKInspector* is an off-device and fully-fledged Android static tool [ERH+16]. APKInspector presents a GUI interface that helps researchers to perform tasks such as presentation of Dalvik bytecode and Java source code, control-flow graphs and call graphs. Finally, *Drozer* is a complete attack and examination framework that is publicly available to everyone [YZW+14]. By using this framework, Android devices could be remotely exploited in order to evaluate their security reliability. Drozer also uses an Agent app on the device and a Python script at the server side as an off-device approach to split the workload of detection and attack.

Off-device and hybrid tools require special knowledge to work. On-device tools face many limitations regarding permission and resources, making them less efficient than hybrid or off-device approaches. Each tool also offers some specific features, and may use different methodologies and deployment methods. For instance, APKInspector only does analysis, and uses the static approach outside of a device, Androguard conducts assessment, analysis and detection via the static approach outside of a device, and Andromaly detects malware via dynamic and profile-based methodologies via the distributed approach.

4 Research Plan

Ongoing threats against users and existence of malicious applications in Google Play store and third-party websites threaten the smartphones and private information of millions of users. At the same time, at least a basic knowledge is required to perform or deal with security assessment tools. This issue leaves ordinary users far behind in understanding what is their current issue. In addition, security tools responsible for detection or prevention of malware on devices have different sorts of limitations. This gap allows malware authors to target unaware or inexperienced users more easily.

Resource usage is a big concern for users to have their phone last longer and perform faster. For this reason, a hybrid approach should be less dependent on the phone and transfer most of the tasks to its counterpart which can be a cloud or a computer. It is also important to an ordinary user to carry a hybrid system which is almost always available to perform detection and assessment phases. As the internet might not be always available, and using limited bandwidth by users could be a big hindrance for a cloud solution, it is acceptable to have a small device in addition to the phone which could be used either at home or work.

Android Debug Bridge (ADB) allows an Android phone to be debugged wirelessly, and this facilitates the process of an external debugger without being connected to a PC or external device by a cable. On the external device, a combination of analysis methods such as network traffic analysis, file operations monitoring, identifying SMS/CALL misuse and data leakage, resource hogger app analysis, logcat and native code analysis is implemented. Unlike many other solutions that can be run on any computers or laptops, we want to propose the external device to be a mini computer like raspberry pi which is small and portable. The device power consumption on a battery while a user is on the way and has no access to any outlet, optimization of all modules and their performance and using limited resources of a mini computer could be all challenging as well as the detection and assessment phase of the system. In case any hint of malware is found, the user can be alerted immediately of the findings regarding the applications installed on his phone.

According to our vision, software security should not be a privilege, but a fundamental building block of future computing. In this spirit, we want to bridge the gap and remove as many as hindrances between ordinary users and detection of malware with the least required knowledge. This should be done via the simplest possible method for users with the least hassle for them to carry or configure the device. Also, the coverage of the detection system should be vast enough to identify any suspicious activity.

5 Conclusion

In this paper, we first described Android's application structure and its security issues. Afterwards, various types of security assessment tools deployment, as well as some examples of current wide-spread tools and frameworks, were reviewed. Eventually, we address the challenge of ordinary users who are not able to work with current security assessment frameworks due to the complexity of configuration, lack of technical knowledge and their limited coverage in terms of features. Then, we propose a hybrid system on a mini computer which is portable, energy efficient and able to perform various common assessment tasks while the user is at work, home or on the way.

Acknowledgements

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Analysis" (SNSF project No. 200020-162352, Jan 1, 2016 - Dec. 30, 2018).

References

[BGM+16] Hamid Bagheri, Joshua Garcia, Sam Malek, Alireza Sadeghi, Hamid Bagheri, Joshua Garcia, and Sam Malek. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Apps. *ISR Technical Report*, 43(6):24, 2016.

[BHM+15] Damjan Buhov, Markus Huber, Georg Merzdovnik, Edgar Weippl, and Vesna Dimitrova. Network Security Challenges in Android Applications. *2015 10th International Conference on Availability, Reliability and Security*, pages 327–332, 2015.

[BK17] Taniya Bhatia and Rishabh Kaushal. Malware detection in android based on dynamic analysis. *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*, pages 1–6, 2017.

[CFGW11] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in Android. *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*, page 239, 2011.

[CT05] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 2005.

[ERH+16] Meisam Eslahi, Mohammad Reza Rostami, H Hashim, N M Tahir, and Maryam Var Naseri. A Data Collection Approach for Mobile Botnet Analysis and Detection A Data Collection Approach for Mobile Botnet Analysis and Detection. (April):199–204, 2016.

[MAC+17] Luca Massarelli, Leonardo Aniello, Claudio Ciccotelli, Leonardo Querzoni, Daniele Ucci, and Roberto Baldoni. Android Malware Family Classification Based on Resource Consumption over Time. pages 31–38, 2017.

[MCW+17] Zhao-hui Ma, Zi-hao Chen, Xin-ming Wang, Rui-hua Nie, Gan-sen Zhao, Jie-chao Wu, and Xue-qi Ren. Shikra: A Behavior-Based Android Malware Detection Framework. *2017 International Conference on Green Informatics (ICGI)*, pages 175–184, 2017.

[Nez17] Maryam Nezhadkamali. Android malware detection based on overlapping of static features. (Iccke):319–325, 2017.

[PFB$^+$15] Malware Penetration, Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, and Manoj Singh Gaur. Android Security : A Survey of Issues , Malware Penetration, and Defenses. 17(2):998–1022, 2015.

[SKE$^+$12] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "Andromaly": A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[sym16] Symantec. Internet Security Threat Report, 2016.

[YZW$^+$14] Kun Yang, Jianwei Zhuge, Yongke Wang, Lujue Zhou, and Haixin Duan. IntentFuzzer: Detecting Capability Leaks of Android Applications. *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, (June 2014):531–536, 2014.

[ZZC17] Lv Zhuo, Guo Zhimin, and Chen Cen. Research on android intent security detection based on machine learning. *Proceedings - 2017 4th International Conference on Information Science and Control Engineering, ICISCE 2017*, pages 569–574, 2017.