2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

Automated Detection and Analysis for Android Ransomware

Tianda Yang, Yu Yang, Kai Qian, Dan Chia-Tien Lo
Department of Computer Science
Kennesaw State University
Marietta, GA, USA
{tyang4, yyang5, kqian, clo}@spsu.edu

Ying Qian
Department of Computer Science
East China Normal University
Shanghai, China
yqian@cs.ecnu.edu

Lixin Tao
Department of Computer Science
Pace University
New York, USA
ltao@pace.edu

***Abstract*—Along with the rapid growth of new science and technology, the functions of smartphones become more and more powerful. Nevertheless, everything has two aspects. Smartphones bring so much convenience for people and also bring the security risks at the same time. Malicious application has become a big threat to the mobile security. Thus, an efficiency security analysis and detection method is important and necessary. Due to attacking of malicious application, user could not use smartphone normally and personal information could be stolen. What is worse, attacking proliferation will impact the healthy growth of the mobile Internet industry. To limit the growing speed of malicious application, the first thing we need to know what malicious application is and how to deal with. Detecting and analyzing their behaviors helps us deeply understand the attacking principle such that we can take effective countermeasures against malicious application. This article describes the basic Android component and manifest, the reason that Android is prevalent and why attacking came in. This paper analyzed and penetrated malicious ransomware which threats mobile security now with our developed automated analysis approach for such mobile malware detection.**

Keywords—Android application analysis, Automatic analysis, static analysis, dynamic analysis

I. Introduction

Android is leapt to 84.4% of Smartphone OS Market Share, Q3 2014[1]. The reason why Android platform is popular can include: (1) Global partnerships and large installed base; (2) Powerful development framework; (3) Open marketplace for distributing your apps [2]. However, there is a hidden crisis behind the prosperous — mobile threat. Mobile threat defined using three categories: malware, chargeware, and adware. In the past year, malware grow substantially in the world. There is an astounding 75% increase in Android mobile malware encounter rates in the United States compared to 2013[3].

There are some reasons caused Android become a famous attacked operating system: (1) the Android fragmentation problem. Android Jelly Bean is now running on more than 40% of Google-flavored devices, but there are many devices running other Android system such as 4.0 Ice Cream Sandwich, 2.3 Gingerbread, 2.2 Froyo etc. [4]. These old editions of Android OS do not have the latest security functions. (2) Android play store has not been popularized in many countries. Apps in the play store are more safety than install non-authorization apps. (3) Globalization, a malicious app can be easily distributed between countries through internet. Many developing countries may suffer from such malware attacks without protection and prevention.

Nevertheless, these above reasons ~~above~~ cannot solve in a short time, so we need some techniques to detect malicious apps and protect our devices. The best technique appropriate for most people is automatic analysis, which people don't have any knowledge of Android can also discriminate between malicious apps and healthy apps. Automatic analysis can be classified ~~split~~ into two of the most popular types of security analysis and evaluation method: static analysis and dynamic analysis.

Static analysis is a thorough white-box type of analysis which inspects an app for its security vulnerability, malicious code, and security threats with respect to data structure and states in a non-runtime environment. It performs the inspection in a more conservative way with data source conservation but it is impossible for a pure static analysis to discover all threats and determine all possible attack details without program execution. Dynamic analysis is usually executed in a runtime sandbox environment to dynamically inspect subtle security vulnerability or part of attack path. However, an exhausting precise evaluation by dynamic analysis for entire program will be impractical.

The analysis ability to evaluate suspected software both statically and dynamically is becoming increasing important. We propose an automatic analysis concept which aggregates the static and dynamic analysis for detecting security threat and attacking in mobile app. The key of this concept is the unification of data states and software execution on the critical test path. In this two-phase approach, a pilot static analysis will first identify the possible attack critical path based on Android API and the existing attack patterns and a dynamic analysis will follow the identified directed path to execute the program in a limited and focused scope to detect the attack possibility by checking conformance of detected path with the existing attack patterns. In the second phase of runtime dynamic analysis, dynamic inspection will report the type of attack scenarios with respect to type of confidential data leakage such as web browser cookie and others without accessing any real critical and protected data sources in mobile device. In this paper, we analyze some popular malware to illustrate the automatic analysis concept.

II. RELATED WORKS

97% of mobile malware is on Android [5]. Because of the tremendous amount of malwares, we need some security and antivirus apps to protect devices. There are top 5 security apps: 360 Security, Avast Mobile Security, ESET Mobile Security & Antivirus, Avira Antivirus Security, and AVL [6].

One of the important aspects of security techniques is to identify and detect malicious hacking and to prevent from hacking attacks. The common hacking techniques include: (1) Third Party App Hacks, which could be anything from a simple wallpaper app to an app that your app allows access to. (2) Cloned Apps which a hacker can distribute the cloned version of your app and any data entered by users on this app is directly stored by hackers. (3) Insecure Wi-Fi which could inject malicious data or apps into your android device through insecure Wi-Fi [7]. However, hacking cannot leave without permissions and system intent. Thus, most security applications try to ~~analysis~~ analyze permissions and information leaking issue to find out ~~beat~~ malicious apps [8]. They are effective for static, but assuming a malicious app run in the background, your sensitive information would still be stolen. To detect running malicious apps, we have to use real time analysis called taint analysis.

There are several taint analysis software include: (1) FlowDroid, it is a context-, flow-, field-, object-sensitive and lifecycle-aware static taint analysis tool for Android applications. FlowDroid analyzes each statement and generates a source-to-sink connection to make sure no taint is lost. [9] (2) TaintDroid extension to the Android mobile-phone platform, ~~that~~ tracks the flow of privacy sensitive data through third-party applications [10]. (3) SCanDroid - ~~, it is~~ a platform for building static analyses on Dalvik bytecode, with specifically focuses on tracking dataflow [11]. Dalvik is the process virtual machine (VM) in Google's Android operating system. The results of taint analysis will necessarily reflect approximating information regarding the information flow characteristics of the system to which it is applied [12]. (4) DroidMiner, a new approach to scalably detect and characterize Android malware through robust and automated learning of fine-grained programming logic and patterns in known malware [13]. The popular ~~majority~~ security applications and platform have been introduced above are focusing on information disclosure, especially for malware. However, there is a new type of malicious and old detection techniques may miss it. It's called ransomware.

III. ANDROID FUNDAMENTAL,SECURITY VULNERABILITY, RANSOMEWARE

Ransomware is a type of malware that locks users out of their mobile devices in a pay-to-unlock-your-device ploy, grew by leaps and bounds as a threat category in 2014 [3]. Ransomware is different from the typical malware which purpose to gain sensitive information and send out to the attacker. Obviously, the traditional security and antivirus applications cannot detect and remove ransomware. In this section, we analyze popular ransomware and ScarePackage to illustrate the principle of ransomware.

A. *Android Fundamental and Security Vulnerability*

Android apps are written in the Java programming language. Android apps have four types of components: (1) Activities, represents a single screen with a user interface. (2) Services, is a component that runs in the background to perform long-running operations or to perform work for remote processes. (3) Content providers, manages a shared set of app data. (4) Broadcast receivers, is a component that responds to system-wide broadcast announcements [14]. Three of the four component types except content providers are activated by an asynchronous message called intent. Android intent binds individual components and associates them with others whether the component belongs to your app or not ~~another~~. The other component type, content provider, is not activated by intents. Rather, it is activated ~~when targeted~~ by a request from a ContentResolver [14].

Besides, android apps have a manifest file to declare all its components. This manifest file also contains user permissions, the app requires and minimum API level. Therefore, manifest file can be thought as a catalog of android application, and it plays a decisive role in android detection and related techniques.

Because of the open source feature of Android, there are several types of vulnerability been found in Android as shown in fig.1, and they are published by CVE (Common Vulnerabilities and Exposures) [15].

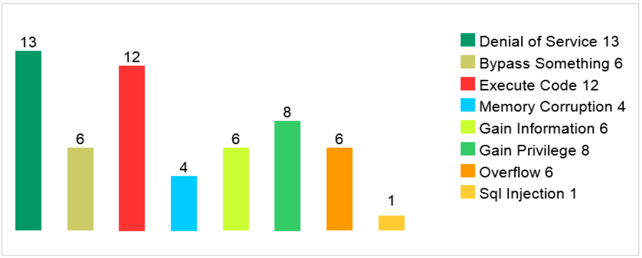

Fig. 1 Types of Vulnerability Collect from 2009 – 2015

Android programmers should avoid such vulnerability during development. Security and antivirus application should pay attention on such vulnerability during detection.

B. Ransomeware "ScarePakcage"

ScarePakage masquerades as an Adobe Flash update or a variety of anti-virus apps, and is distributed as a drive-by-download. When downloaded, it pretends to scan victims' phones and then lock~~s~~ the device after falsely reporting that its scan found illicit content. ScarePakage then displays a fake message from the FBI and attempts to coerce victims into paying them to avoid criminal charges and regain control of their device [16]. Your device cannot do anything until you pay to the attacker to unlock device, the payment method in this case is MoneyPack.

The successful of ScarePackage is not only using new hacking techniques but also taking advantage of people's fears and concessions to avoid troubles. No matter how successfully it does, we have to detect and remove it from our device. The main function of ScarePackage is to lock device screen. Let's see the necessary permissions and components related to locking function.

DeviceAdminReceiver and DevicePolicyManager are two major classes we need to pay attention to. DeviceAdminReceiver is an onReceive() method return operations based on different actions. DevicePolicyManager is public interface for managing policies enforced on a device, especially for locker. Most clients of this class must have published a DeviceAdminReceiver that the user has currently enabled. The necessary permission is shown in Fig. 2.

```
android:permission="android.permission.BIND_DEVI
CE_ADMIN"
<intent-filter>
<action
android:name="android.app.action.DEVICE_ADMIN
_ENABLED" />
<action
android:name="android.app.action.DEVICE_ADMIN
_DISABLED" />
</intent-filter>
```

Fig. 2 Permissions for locking

Besides, ScarePackage has other functions to support locking function. Using a Java TimerTask, which is set to run every 10 milliseconds, the application will kill any other running processes that the user interacts that are not the malware itself or the phone's settings application. This TimerTask is run in a background service and the necessary permissions and components are shown in Fig. 3.

```
<uses-permission
android:name="android.permission.GET_TASKS" />
<uses-permission
android:name="android.permission.KILL_BACKGRO
UND_PROCESSES" />
```

Fig. 3 Permissions for killing process

In the same service, it uses an Android WakeLock to prevent the device from going to sleep. The necessary permissions are shown in Fig. 4.

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

Fig. 4 Permissions for WakeLock

At the meantime, ScarePackage has a boot receiver to run the application during the device starting. The necessary permissions are shown in Fig. 5.

```
android:permission="android.permission.RECEIVE_
BOOT_COMPLETED"
<intent-filter>
<action
android:name="android.intent.action.BOOT_COMPL
ETED" />
<category
android:name="android.intent.category.HOME" />
<action
android:name="android.intent.action.SCREEN_ON"
/>
</intent-filter>
```

Fig. 5 Permissions for Boot Receiver

When victims paid them, ScarePackage would uninstall itself. The paying method goes through the internet. The necessary permission is shown in Fig. 6.

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

Fig. 6 Permissions for Paying Function

Ransomware has special permissions and components. Security apps do not have an efficient way to detect and remove ransomware. Therefore, we need an automatic analysis tool to distinguish it from health application.

IV. DESIGN OF AUTOMATED ANALYSIS

What is automated analysis? This question doesn't make any sense, because different people with different background may have different answers. Nevertheless, the principle of automatic analysis may tend to a same point — better detection. There is a high performance protection system around us called immune system, which is a system of biological structures. Immune system protects against disease. In this system, it can detect a wide variety of pathogens and distinguish them from healthy tissue. However, immune system has a weakness that cannot be solved. It is a passive working mode. In Android detection tool, it can only detect static malicious apps. The ideal security detection tool must contain both passive and active mode. Based on the above system, we can start our security analysis for Android.

When an APK came in, the first part is static analysis. Static analysis includes:

(1) APK De-compilation: decompile APK into .dex file, .xml files, manifest files, and resource files.

(2) Repackaging Detection: APKs can be repackaged by an unknown third party and injected malicious resources in it. Additional step is needed to decode the repackaged code.

(3)Features Extraction and Comparison: search manifest file and compare with features database to detect whether this app is malicious or not.

However, static analysis cannot deal with the situation such as code obfuscation and encryption, so we need dynamic analysis to help us. Dynamic analysis is needed to monitor the behaviors of the running application and detect whether the behaviors match the malicious activity or not, the flowchart of the automatic analysis is shown in Fig. 6.

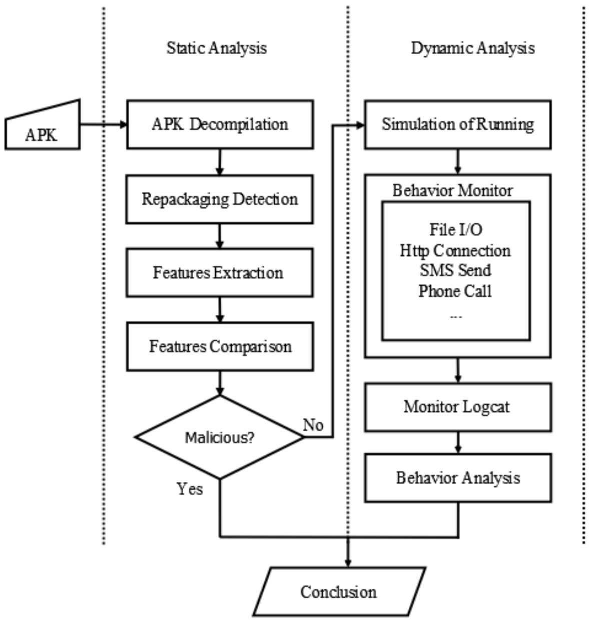

Fig. 6 Automatic Analysis Overview

A. Malicious Features in Static Analysis

Static analysis is to analysis the APK file. APK file is actually a zip compression package and can be decompressed. The structure of a decompressed APK file is shown in Fig. 7.

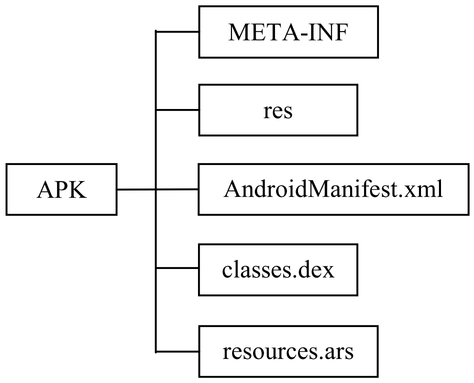

Fig. 7 Structure of APK File

An APK file contains much information. META-INF directory stores signature data and is used to ensure the integrality of APK package. Res directory is used to store resource files such as images and UI layout. Manifest file describes name, version, permissions, library files, and other information. Classes.dex is a java byte file and can be decompiled to java or other programming language. Resource is a binary resource file after compilation.

We can obtain useful information from APK file to detect malicious apps. Malicious apps have different types. Some of them use similar hacking techniques and we called a malicious family. A malicious family may have several malicious features. Static analysis is based on features matching. In this paper we pick four parts to illustrate features in static analysis.

(1) Permissions:

Android system has its own permission management mechanism. Many functions such as user experience and data dependency need related permissions to access. Malicious application also needs permissions to execute malicious functions. For examples, some malware use SMS to send out sensitive data and this malware contains SEND_SMS permission in manifest file. However, some communication applications such as Whatsapp may also contain SEND_SMS and READ_SMS permissions. Therefore, we cannot simply make a decision just based on permissions only.

(2) The sequence of API invoking:

Some malwares have similar hacking techniques and their method invoking may be similar. For example, using SMS to send out data may have similar method invoking, the sequence is OnCreat → sendmsms → sendTextMessage. We can decompile classes.dex to smali and recall the sensitive API to extract the sequence of method invoking.

(3) Resources:

Some Android applications have executable files that are encrypted to disguise into ~~become~~ resources files and the encryption executable file will be executed when the malware is running. The application DroidKungFu is a malware using this technique. We can find a ratc file belongs to asserts directory in the extraction APK file, and this ratc file is encrypted but will be decrypted at runtime to get the administrator privileges.

(4) APK structure:

Because of the specificity of APK structure, APKs would be easily repackaged and released. Many attackers use efficient and automatic way to inject malicious functions into normal applications. We can use package name and signature and comparing with the official application to verify repackaging.

B. Malicious Features in Dynamic Analysis

Dynamic features may be sophisticated and we collected four behaviors are shown in TABLE I.

TABLE I. BEHAVIORS OF DYNAMIC ANALYSIS

#	*Behaviors Name*	*Behaviors description*
1	Critical Path and Data Flow	Command Execution and Sensitive Data Access
2	Malicious Domain Access	HTTP Server Access
3	Malicious Charges	Charge Without Clear Notification
4	Bypassing the Android Permission	Execution Unauthorized Activities

(1) Critical Path and Data Flow

Android system is based on Linux kernel, and there are sensitive paths such as system executable directory /system/xbin, malicious app could invoke system functions under the directory. Some malware can get administration privileges and execute chmod, chown, mount and other commands to change the permission of files and directories. SMS, contact info, and other sensitive data store in a specific database. The sensitive data paths and data information are shown in TABLE II.

TABLE II. SENSITIVE PATHS AND DATA DESCRIPTION

Sensitive Paths and Data	*Description*
/system/xbin	System executable directory
/system/bin	System executable directory
/data/system/accounts.db	Personal Account Information
/data/data/com.android.providers.contacts/databases/contacts2.db	Contact Information Database
/data/data/com.android.providers.telephony/database/mmsms.db	SMS Database

Malware can also access to Android file system and using common APIs to operate files, the common APIs are shown in TABLE III.

TABLE III. APIS FOR I/O OPERATION

APIs	*Description*
Public static File creatSDFile(String path, String fileName)	Create file in SD card
Public static File creatSDDir (String dirName)	Create directory in SD card
Public Boolean isFileExist(String fileName)	Whether the file exists or not
Public static File write2SDFromInput(String filename, InputStream input, int len, String dirName)	Write the data of InputStream to SD card

(2) Malicious Domain Access

Stolen data by malware would send user's information to a specific domain. Zombie and Trojans malwares would send to C&C server to get commands. We can collect those malicious domain names to build a blacklist as a factor to distinguish malwares. We can also track the data flow to monitor the sending out data. The processes for HTTP uploading and downloading are shown in TABLE IV.

TABLE IV. DATA FLOW FOR HTTP

Function	*Process*
Upload	1. Get connection using URL.openConnection. 2. Set request method to post using HttpURLConnection.setRequestMethod("POST") 3. Write data to output stream using HttpURLConnection.getOutputStream() 4. Get response code using HttpURLConnection.getResponseCode() 5. Disconnect using HttpURLConnection.disconnect()
Download	1. Get connection using URL.openConnection. 2. Get input stream using HttpURLConnection.getInputStream() 3. Disconnect using HttpURLConnection.disconnect()

(3) Malicious Charges

An App is charged for a service without clear notification is called chargeware. Chargeware may use charge SMS or Call to pay for services. In dynamic analysis, we can track the destination of SMS and Call. If there is an unknown destination, the software would be considered as a malware. Besides, ransomware also allowed user to pay but it use HttpPost to post MoneyPack number to specific server.

(4) Bypassing the Android Permission

If an application does not announce some permissions, but the application execute some related work with those permissions. It's called bypassing permission. This situation takes place when an android application gets administration privileges. If an application has administration privileges, it could execute sensitive operation without any related permissions.

V. CONCLUSION AND FUTURE WORK

This paper penetrated and analyzed the characteristics of ransomware and introduced different taint tools. Regardless of what techniques we use, the main goal is to build a better performance tool. In this paper, we illustrated a design of automatic analysis. Even though it does not tell you how to implement a completed automatic tools, but helps to understand what should do to approach better detection.

The future research could focus on the combination of static patterns and the APIs collection of dynamic analysis. New types of malicious application would grow up along with the new technology and new requirement.

VI. ACKNOWLEDGMENTS

The work is partially supported by the National Science Foundation under award: NSF proposal #1244697, #1438858, #1241651. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[2] http://developer.android.com/about/index.html

[3] https://www.lookout.com/resources/reports/mobile-threat-report

[4] http://securitywatch.pcmag.com/android/308966-android-s-biggest-security-threat-os-fragmentation

[5] http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easy-way-you-stay-safe/

[6] http://www.digitaltrends.com/mobile/top-android-security-apps/

[7] https://appvigil.co/blog/how-to-common-techniques-used-for-hacking-android-apps/

[8] Suzanna Schmeelk, Static Analysis Techniques Used in Android Application Security Analysis, December 1, 2014

[9] http://sseblog.ec-spride.de/tools/flowdroid/

[10] W. Enck, P. Gilbert, B. gon Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. *TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones*. In OSDI, pages 393–407, 2010.

[11] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. *SCanDroid: Automated Security Certification of Android Applications*. Technical Report CS-TR-4991, Department of Computer Science, University of Maryland, College Park, November 2009.

[12] T. Terauchi and A. Aiken. "Secure information flow as a safety problem". In 12th International Static Analysis Symposium, September 2005.

[13] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, P. Porras. DroidMiner: Automated Mining and Characterization of Fine-grained Malicious. TechReport, 2014

[14] http://developer.android.com/guide/components/fundamentals.html

[15] http://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

[16] https://blog.lookout.com/blog/2014/07/16/scarepakage/