IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Investigating and Analyzing Malicious Events in Android Application

Aparna Chandran

Department of Computer Science and Engineering, Nehru College of Engineering and Research Centre, Thiruvilwamala, Thrissur, Kerala, India

aparnachandran1188@gmail.com

Abstract

Smart mobile devices have been widely used and the contained sensitive information is endangered by malware events and codes. The malicious events caused by malwares are crucial evidences for digital forensic analysis, and the main task of mobile forensic analysis is to find the malicious codes and reconstruct these events. However, the reconstruction heavily relies on the code analysis of the malware. The difficulties and challenges include how to quickly find the suspicious programs, how to remove the anti-forensics tricks of malicious code, and how to deduce the malicious behaviors according to the code. To address this issue, a systematic procedure of analyzing typical malware behaviors on the popular mobile operating system Android is proposed. Based on the procedures, the deduction of Android malicious events is also discussed.

Keywords: Antiforensics, obfuscation, deobfuscation.

Introduction

Mobile devices are nowadays widely used to deal with sensitive personal affairs, and are becoming an attractive platform for cybercriminals. With the fast evolvement of mobile OS such as Android and iOS, and the growing processing capability of mobile hardware, the number of smart mobile devices grows exponentially. A huge number of malwares are developed to threaten the data privacy and system security of smart mobile devices, and bring new challenges to forensic analysts. A modern digital forensic analyst should know these threats and be able to employ forensic analysis against mobile malware. Most of the forensic analysis on mobile devices focuses on the data acquisition process. However, the scope of mobile malware forensics extends from simple information retrieval to a series of events reconstruction. The mobile malware forensics often involves four aspects, identification of suspicious programs, defeating the antiforensics code, extracting malicious code from malwares and malicious functions deduction. Traditional forensics discusses the process of collecting static data as digital evidence. The reconstruction of malicious events involves connecting the relationship between programs, operating system, hardware and I/O data. Tiny details may be main obstacles of malicious events reconstruction. Modern mobile malwares are designed towards certain platform. The complexity of both hardware (different CPU architectures, file systems) and software (new mobile operating system) challenge the inexperienced analysts. The architecture and design pattern of mobile applications differ widely from common applications on personal computers. The purpose of this paper is to present a systematic process of Android malware forensic analysis, focusing on the deduction and reconstruction of malicious events.

Android Malware

Most of the malwares on Android OS are developed using JAVA programming language and are executed on Dalvik VM engine of the system. Although Android itself is a Linux based system, the best way of malware invasion is via normal application installation. Thus to analyze malwares on Android OS, the analyst should first understand the format of the Dalvik VM based program. The Dalvik Based Android applications are released and stored in the device with the APK format. An application is first compiled and is then archived into one single APK file with all of its parts, including codes and assets. The APK file is actually an application in the form of a ZIP archive with codes, resources, assets, certificates and manifest file. The inner folders and files structure of this archive conform to the JAR file format specification. After the installation, the APK file is copied to a specific location in the system. For system applications, the location is typically /system/app and for user installed applications the location is /data/app. From the forensic analyst's point of view, an APK file contains three parts of abstract information: signature, bytecodes and resources.

The signature contains the message digest of the APK file. Since any modification to the APK file will change the message digest of the signature, one could quickly identify if an application is corrupted by checking the signature. Analyst could also collect signatures of malwares to find out malwares quickly. The executable part of the application, the classes.dex file in the archive, contains all compiled classes of the program in the form of bytecodes. For Android programs, the original JAVA bytecodes are converted to the instruction set used by the Dalvik VM, which is a register-based VM while JVM is stack-based. Resources is the non-executable part of the application, it contains all additional data required by the application. Most resources in an application are user interface components, such as bitmaps, menus, layouts, widgets. In most cases, the malicious part of the malware runs in background and does not have any user interfaces. So these UI resources are seldom concerned. However, the resource file, AndroidManifest.xml, is important that indicates crucial forensic information of an application.

The AndroidManifest.xml file is encoded into binary format in the APK file. It contains the permission request of an application. The most important forensic information are permissions and components. In order to access some protected APIs of Android, the application will declare the permission request in AndroidManifest.xml, such as the permissions to read message, contacts, etc. Permission request is a very important clue to reveal malicious functions. For instance, a normal application, such as calculator, declares a READCONTACTS permission, it can be very suspicious because a calculator should never need information about contacts. This character is unique for Android applications and is useful for analysis. Android applications are formed by components. The components of the application are divided into four kinds – activities, services, broadcast receivers and content providers. A malware who runs in background often has a service component and a receiver component in order to receive the boot Intent on system booting. By checking components and the received intents, analyst may have a brief view of the potential behavior of an application.

Identification of Suspicious Application

In a typical smart mobile device there are as much as hundreds of applications. Malwares only occupy a few parts of applications and most of the others are benign. The first step of forensic analysis is to identify the malicious programs from the benign ones. Although many research works and tools are claimed to support malware detection, there are still some unsolved problems for forensics. In one way, automatic tools need samples to generate malware database. The rapid evolvement of malware makes automatic detection tools difficult to follow [3]. Moreover, some malwares are designed for attacking specific devices and yet are hard to be collected beforehand. In another way, forensics not only needs to find the suspicious programs, but also requires code analysis and events reconstruction. Thus manual check is helpful for later in-depth analysis, and manual methods are essential for forensic analyst to ensure the identification. To identify malicious programs, one important conclusion is that malwares are always connected with some unusual features. These features indicate the potential suspicions.

For excluding benign applications from affected ones, the message digest is a useful cryptographic feature. A database for normal applications can be built by collecting message digest information from online markets. Then the analyst simply checks an application’s message digest and if the message digest of the checked application cannot be found in database, it is possible that this application is malign. However, only with the message digest it is not cautious to determine the malicious applications. A more in-depth analysis should be employed to fulfill the identification.

The permission requirement is a unique character for Android programs. Due to the design philosophy of the Android OS, the application only needs to apply for permissions when being installed and persistently own these permissions without repeatedly requesting. Users may ignored the initial request, and a common malware pretending to be an unharmful application with faked normal functions will ask for a set of permissions such as SMS and Contacts database access, even the faked functions of the application need not these permissions at all. Suspicious permission requirement is the leading clue to confirm an Android malware. Most of the malwares declare a list of high-privilege permissions to fulfill malicious functions.

At the very abstract level, Android application is formed by components. The structure of components can be used to judge the program’s characters. The service component and receiver component are sensitive weapons for malwares. So, from the examination of components and the received intents, analyst could have a brief view of the potential malicious function of an application and suspicious applications are to be distinguished from the normal ones.

Anti-forensics Techniques

Events could be deduced from the code. However, malware developers always try to stop the deduction or make it hard. Before code analysis, one important thing is to clean the barrier – anti-forensics codes. Anti-forensics codes are common inside malwares of commodity personal computers. For instance, many malwares detect the execution environment to check whether it is executed inside a virtual machine. Android malwares inherit the property to inconvenience the forensic analysis. The three common anti-forensics techniques are obfuscation, string encryption and environment verification.

Obfuscation

The obfuscation techniques of Android malware are as much the same as JAVA obfuscation, because the developing programming languages are similar. A very typical case is that in an obfuscated program all of the packages, classes, methods, fields are renamed to single alphabet such as a, b, c.a(), d, e.b, f.a, g.b(). So that analyst is hard to distinguish different parts of the code yet is difficult for her to understand the functionalities [4].

Strings Encryption

For an experienced reverse engineer, strings in a program are valuable information sources. Many malwares use string encryption to avoid plaintext detection. Constant strings in malware are encrypted with symmetric algorithms such as DES and the AES and the key is fixed (dynamic key is seldom used because no matter how complex the key is, it will finally be used to decrypt the ciphertext). The encryption makes static analysis hard. However, if the analyst has the capability of dynamic execution, the analyst may manually extract key and decrypt the ciphertext, thus the information is still available for retrieving.

Environment Verification

Some of the mobile malwares are designed to attack certain types of mobile devices. Specific symbols like Android system properties (from android.os.BUILD) are often verified to make sure the malware is not executed in an emulator or other types of devices. And the subscriber ID (IMSI) is used to make sure the malware is running on a certain device with the special IMSI. If verification fails, the malicious code will stop executing, and the analyzers could not simply reproduce the malicious behavior by emulation or using any improper devices. This anti-forensic technique lets malware deceive dynamic black-box analysis.

Defeat Anti-forensics codes

Some countermeasures to the anti-forensics techniques mentioned above are decompilation and deobfuscation, strings decryption and program patching.

Decompilation and Deobfuscation

For an Android application, the high level JAVA-like source code is much easier to read and to be understood than the bytecode. However, the State of Art decompilation tools cannot decompile programs perfectly. The decompiled source code typically contains mistakes or code absences. The bytecode is always correct and accurate although it is much more difficult to be analyzed. So analyst should utilize both bytecode and decompiled source code, and take both codes into analysis to compensate the shortcomings of each other. The three main steps suggested to employ decompilation and deobfuscation are the analyst could use apktool to extract the bytecode (with .dex format), the combination of dex2jar and jd-gui are helpful to decompile the bytecode file to JAVA source code and the decompiled JAVA source code may contain huge number of errors. The possible options for code fixing are removing empty classes, renaming, decompile errors correction, control flow error correction, name conflict correction and missed information fixing.

Strings Decryption

Strings are important information sources and most constant strings(e.g. remote server URL) in malware are encrypted. Often a decryption process is required to extract these strings. The whole decryption process involves encryption algorithm recognition, secret key extraction and string decryption. One convenient aspect is that many malwares use system cryptographic APIs to deal with encryption and decryption. Analyst could filter out these situations and quick identify the key.

Program Patching

To deceive dynamic analysis, system properties and the subscriber ID are often verified by the malware. In order to employ dynamic analysis, analyst could automatically search for these features and manually patch the code to avoid these verifications.

Detecting the Malicious Events

The core part of mobile malware forensics is to reconstruct the malicious events via program code and additional information such as network flow. But in most cases the only form of malware provided is binary program. In order to understand the logic of the program, a reverse code analysis is essential. Although there is not a standard procedure for reverse code analysis, on Android some typical behaviors may be the key that helps analyst unlock the puzzle and understand the crime. The analyst can follow the patterns to locate and then find the malicious behaviors, and finally combines the behaviors to deduce the events.

Android malwares are written using JAVA programming language, and the bytecode of the malware contains all logic functions. In mobile malware forensic analysis, the direct evidence of malicious events is from the malicious code itself. A malware sample may be acquired after the crime [5]. The criminal events are unknown for analyst. Only the code related to malicious behaviors helps analyst fast locating and analyzing the malicious event first. The reverse code analysis aims at extracting program fragments first, analyst could then combine simple functions into a high-level abstract events. The reconstructed events may include following information, the work flow of malicious code, sensitive information that the malware accessed, the encryption algorithms, and the details of malware's communication protocols. The Figure.1. shows Malicious Events Reconstruction.

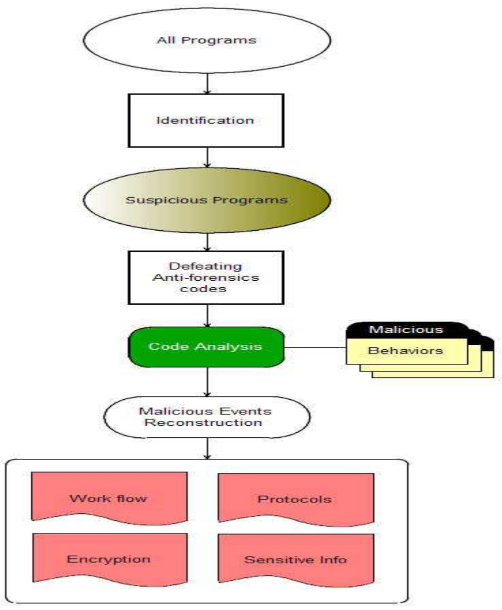

Figure 1. Malicious Events Reconstruction

The task of reconstruction and combination requires not only the mining of function inside codes, but also rearrangement of these functions into a correct order. Android provides a logcat mechanism to capture high-level operations such as the system API calls and services starting/stopping log. If allowed, analyst should try to reappear the execution of malware and record the occurred operations, and then draw the picture of the events.

Case Study

A complete forensic analyzing process to show some details of mobile malware analysis is described below. The process of forensic analysis could be divided into four parts. The Figure 2. shows The Suspicious Application Utility Hub.

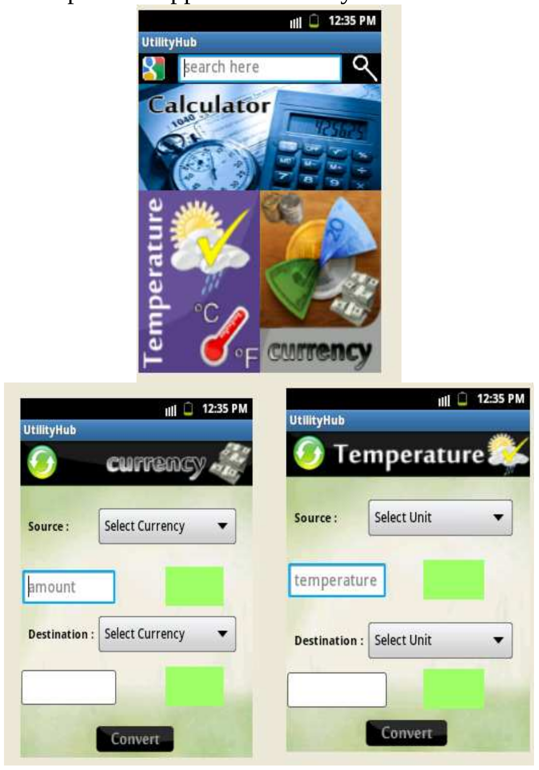

Figure 2. The Suspicious Application - UtilityHub

Background

The challenge offers the exploration of a real smart phone compromised by mobile malware, based on Android, after a security incident. Analyst will have to analyze the image of a portion of the file system, extract all that may look suspicious, analyze the threat and finally give conclusion.

Identification of Suspicious Program

There were totally ten applications and two tmp files contained in the provided corrupted memory dump. Among the ten applications, seven files' message digest can be found online and are considered as normal, trusted packages.

• com.adobe.reader-1.apk
• com.google.android.stardroid-1.apk
• com.rovio.angrybirds-1.apk
• com.android.vending-1.apk
• com.google.android.apps.maps-1.apk
• com.google.earth-1.apk
• com.opera.browser-1.apk

For the rest three packages, although the message digest are not found online, further examined the permission and component and decompiled these packages, found nothing suspicious for com.google.android.apps.finance-1.apk and net.xelnaga.exchanger-1.apk. Then the only left application, app/com.aditya.utilityhub.apk is exactly the same as the tmp files app/vmdl34052.tmp and lgdrm/TRYSYNC, which means this application was possibly active when the memory was dumped and further checked the requested permission of this application and found the following permissions are requested.

• android.permission.INTERNET
• android.permission.READ PHONE STATE
• android.permission.ACCESS NETWORK STATE
• android.permission.RECEIVE BOOT COMPLETED
• android.permission.VIBRATE
• android.permission.ACCESS COARSE LOCATION
• android.permission.ACCESS FINE LOCATION
• android.permission.CALL PHONE
• android.permission.SEND SMS
•android.permission.READ CONTACTS
• android.permission.RECEIVE SMS
• android.permission.READ SMS
• android.permission.WRITE SMS

The application however performed as a normal 3 in 1 utility tool with Google Search when executed. The unusual number and type of unnecessary requested permissions makes it suspicious and then focused on this suspicious application com.aditya.utilityhub.apk to employ analysis in depth.

Anti Anti-forensics

This malware uses all the three anti-forensics techniques such as obfuscation, string encryption and environment verification to interfere forensic analysis. Here the code error fixing is employed to get a neat version of decompiled code for static analysis. The raw code from the decompiler contains lots of errors. To fix these errors, the bytecode is analyzed, and rebuilt the source code. Since all malicious parts of the code are obfuscated by name renaming (e.g., com.aditya.utilityhub.daemon.g.a.a), it is need to do code refactoring. The Depth-First Search is used in the procedure of refactoring all source codes. The atomic functions were refactored first then the complex ones. When looking into the decompiled code, some empty JAVA classes were found without fields or method definitions. Some of these situations are due to the decompiler's processing capability and the correct code should be manually added after further examination to the corresponding bytecode.

Evidences and Malicious Event Rebuild

According to the code analysis result, rearrange the sequence of each functions and formed a complete malicious communication process. The process contains four parts.

1. Key exchange: The first step of the communication between the malware and the server is a self-defined diffiehellman key exchanging to establish a secret key. The typical Diffie hellman key exchange algorithm is used in the negotiation, and then DES is used in encrypted communication.
2. Encrypted private information sending: The communication between malware and remote server is based on HTTP protocol. The following private information are sent via encrypted communication.
    - device information
    - personal information SMS
    - contacts
    - .daemon.fc9
    - com. aditya.utilityhub.daemon.CCcomServic e
    - com. aditya.utilityhub.daemon.BootReceiver
3. Server command receiving: The malware requests command from server every 15 seconds, and executes the command. Some commands involve extra communication while executing. For commands "getsms", "getcontacts", the malware will send SMS or contacts information to server when executing.
4. The ”smsspy” communication: A very special command from the server is the ”smsspy” command. When receiving, the malware will change the malicious mode into an ”smsspy” mode and send any SMS to server whenever an SMS is coming from mobile network.

Conclusion

In this paper the problem of Android mobile malware forensic analysis is discussed. A huge number of malwares are developed to threaten the data privacy and system security of smart mobile devices. A modern digital forensic analyst should know these threats and be able to employ forensic analysis against mobile malware. The core task of mobile malware forensic analysis is to reconstruct the malicious events according to malware. To address this issue, a systematic procedure of analyzing typical malware behaviors on the popular mobile operating system Android is proposed and based on the procedures, the deduction of Android malicious events are also done.

Acknowledgment

This work is supported by my research guide. I am very thankful to my research guide Mr. Arundas K.V., Lead Android Programmer, BlackJack Apps, Trivandrum, for his support and guidance.

References

[1] Takamasa I., K. Takemori and A. Kubota, "Kernel-based Behavior Analysis for Android Malware Detection," Proceedings of the Seventh International Conference on Computational Intelligence and Security, Berkeley, US, pp.1011-1015, 2011.

[2] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," Proceedings of the 19th Annual Network and Distributed System Security Symposium, Vienna, Austria, pp.129-132, 2012.

[3] Asaf Shabtai, "Malware Detection on Mobile Devices," Proceedings of the Eleventh International Conference on Mobile Data Management, Columbus, US, pp.289-290, 2010.

[4] Ilsun You and Kangbin Yim, "Malware Obfuscation Techniques: A Brief Survey" Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, Berkeley, US, pp.297-300, 2010.

[5] G. Benats, A. Bandara, Y. Yu, J. Colin, and B. Nuseibeh," PrimAndroid: Privacy Policy Modelling and Analysis for Android Applications" Proceedings of the 2011 IEEE International Symposium on Policies for Distributed Systems and Networks, Columbus, US, pp.129-132,2011.

AUTHORS PROFILE

Aparna Chandran, received her B.Tech degree from Calicut university and currently pursuing M.Tech in Computer Science and Engineering at Nehru College of Engineering and Research Centre, Thrissur (Dist.), Kerala, India. Are of interests are Networking, Operating System, Database and Digital Image Processing.