Research on Android Application Reinforcement Method for Mobile Medical Service

Shuwang Zhou¹²³*, Shumei Zhang⁴, Minglei Shu¹²³, Jiyong Xu¹²³, Zhaoyang Liu¹²³

¹ Qilu University of Technology (Shandong Academy of Sciences), Jinan, Shandong 250000, China;
² Shandong Computer Science Center (National Supercomputer Center in Jinan), Jinan, Shandong 250000, China;
³ Key Laboratory of Medical Artificial Intelligence, Jinan, Shandong 250000, China
⁴ The 8th People’s Hospital of Jinan, Jinan, Shandong 250000, China
*Corresponding author’s e-mail: zhoushw@sdas.org

Abstract. In the field of mobile medicine, in order to improve the security of Android applications and prevent Android APP from malicious decompilation and tampering, a security reinforcement scheme for Android APP based on random confusion is proposed. By confusing a large number of random characters in the DEX header file, without affecting the execution efficiency of Android software, it increases the difficulty of decompiling the DEX file, improves the security of the source code, and ensures the correctness and integrity of the DEX file.

1. Introduction

Android is an intelligent mobile terminal open source operating system developed by Google. Since the first Android mobile was released in October 2008, the global market share of Android phones has reached 86.2%. The application of Android has integrated into all aspects of our life. Especially in mobile medical treatment, the application of Android is playing an important role. According to statistics, there are nearly ten thousand kinds of Android mobile medical APP worldwide. However, the openness of the Android system has brought a lot of security problems, and Android APP faces serious threats of decompiling, cracking, adding malicious code, virus implantation, advertisement implantation and so on [1]. Since Android mobile medical APP involves personal data, health files, health assessment, disease warning, treatment plans and other important privacy information, once it is tampered, it will threaten personal health and even life. Therefore, this paper specially designed a comprehensive strengthening scheme for Android mobile medical APP.

2. Threats to Android APK

2.1. APK file structure

The APK file is the installation software of the Android APP, which is essentially a ZIP format compression package, which can be decompressed by compression software and the file structure after unzip is like table 1[2].
Table 1. Apk file structure.

| Catalog          | Effect                                                                                           |
|------------------|--------------------------------------------------------------------------------------------------|
| Asset            | It is used to store static files that need to be packaged into APK.                             |
| Lib              | It is used to store application dependent native library files, where the Lib library contains 4 different types, which can be divided into ARM, ARM-v7a, MIPS, and X86 based on the different CPU models. |
| META-INF          | It is used to preserve the signature information of the application. Signature information can verify the integrity of the APK file and ensure that every file in APK is not tampered with. |
| Res              | It is used to store resource files, and all files in this folder will be mapped to the R file of Android project. |
| Android-Manifest.xml | It is a configuration file for Android application, which configures the components, permissions, and other information necessary for the operation of the program. |
| Classes.dex      | The DX tool removes the contents of each .class file, then reconstructs, and finally generates the optimized DEX file, and the DEX file can be executed on the Dalvik virtual machine, and the speed is faster. |
| Resources.arsc    | It is a compiled binary resource file.                                                            |

The attacker usually understands the execution flow of the program through decompilation, tampering, etc. or directly obtains the source code of the program, modifies the logic code in the running of the program, bypasses the verification protection mechanism of the program, and realizes the purpose of cracking the program[3].

2.2. Decompile
Decompile refers to the attacker's use of disassembly tools or decompilation tools to translate binary executable files into assembly code or program source code, and then attack the malicious code by attacking the program[4].

At present, common apk decomposition tools include jadx, JEB, Android Killer, etc[5]. They can directly decompile Dalvik bytecode directly into Java source code, with high code repurchase and support for cross-indexing, but do not support variables and methods. Rename and confuse code analysis and more. In addition, the attacker also commonly uses apktool, dex2jar, jd-gui three software to achieve the decompilation of the APK, apktool decompres the apk file, dex2jar will decompress the .dex file of the apk file into a jar form, and then use jd-Gui browse the source files of the jar package[6].

2.3. Malevolent tampering
Malicious tampering mainly refers to the malicious modification of the program execution process or source code obtained by an attacker, such as shielding the original program's verification and protection mechanism, adding malicious code, adding advertising links and so on. According to the tamper, the program tampering can be divided into two types: static tampering and dynamic tampering[7].

Static tampering refers to tampering client behavior by modifying client files before the program runs. However, when the Apk file is installed, the system will compare its signature information and determine the integrity of the program, so as to decide whether the Apk file can be installed. So this kind of tampering needs to bypass the verification of the integrity of the program, and then use apktool and other tools to repackage the modified application.

Dynamic tampering refers to modifying the application data in the memory while the program is running, and the game memory modifier is common. It is difficult to implement dynamic tampering under the platform.
Classes.dex is the core file of apk. The Java source code is compiled into a .class file, which is then converted to classes.dex by the dx tool, which runs on the Android Dalvik virtual machine. Therefore, the classes.dex file is the key target of the tamper[8].

First, the tamper uses baksmali to process classes.dex as a Smali file, then convert it to a jar file with des2jar, then check the code with JD-GUI, determine the position to be modified, find the corresponding modification position in Smali, and then recompile it to classes.dex text with Smali, overwrite the original classes.dex file, and recheck it. The tamper complete the attack[9] after re-signing.

3. Design of reinforcement scheme

This paper specially designed the confusion encryption scheme of classes.dex file for mobile medical APP, which realized the reinforcement of Android APK, increased the difficulty of malicious attacker decompiling and cracking APP, and protected the security of user privacy data.

This paper implements the security hardening of Android software from two aspects. The first is the random confusion and reinforcement of the dex header file, and the second is the secure dynamic loading and reinforcement of the bin file. The Android software is protected by a random confusion collaborative reinforcement method, which enhances the security of the code. At the same time, the integrity of the APK file is guaranteed by the hash value. The random confusion reinforcement process is as follows, and flowchart is shown in Figure 1.

1) Unzip the Android software that needs to be hardened, and get the dex file and the bin file respectively.
2) Generate a random number 1500<N<2500 using an integer random function.
3) Using the random number N in (2) as a parameter, input a random character generation function to generate a byte array of length length=N.
4) Insert the byte array of length N into the P position of the dex header file. The formula for calculating P is as follows:
   \[ P = 0x28 + N \% (0x70 - 0x28) \]

Figure 1 APK safety reinforcement overall flow chart
Where 0x70 is the original length of the dex header file, and 0x28 is the length sum of the magic, checksum, signature, fileSize, and headerSize 5 fields.

After the confusing byte array is inserted, make sure that the first 5 fields of the dex file are not affected.

5) Customize the random number N, encrypt the result as 8 byte encoding, and append the 8 byte encoding to the end of the dex file.

6) Update the dex file checksum field checksum, which is used to verify the correctness of the files except the magic and checksum fields according to the alder32 algorithm.

7) Update the dex file hash value field signature, signature is to use the SHA-1 algorithm to calculate the dex file to remove the hash, checksum and signature three fields, used to determine file integrity.

8) Update the value of the dex file length field fileSize:
   fileSize=filesize+N+8

9) Update the value of the header of the dex header file length field:
   headerSize=headerSize+N+8

The dex file headers before and after confusion are compared as shown in Figure 2. By confusing a large number of random characters on the dex header file, the difficulty of decompiling the dex file is increased without affecting the execution efficiency of the Android software, the security of the source code is improved, and the correctness and integrity of the dex file are ensured.

![Figure 2 DEX file header contrast before and after confusion](image)

Figure 2 DEX file header contrast before and after confusion

The traditional way of calling

```
classes.dex Call ➔ classbin
```

Using loader to call security

```
classes.dex Call ➔ DexClassLoder Load ➔ classbin
```

Figure 3 the security call of bin file.
The obfuscated DEX file is loaded into memory using the loader DexClassLoder[10], which is a class loader that can load classes from a .jar or .apk file containing classes.dex, and can be used to implement dynamic loading of dex, code hot update, and so on. The headerSize field in the DEX header file is read by DexClassLoder, and the DEX header file is read by the headerSize value. The 8-byte code is read from the tail of the dex file, and the random confusion field length N is obtained by a custom decryption algorithm, and the insertion position P of the random confusion field is calculated by the formula in 4). Remove the Pth to the P+N-1 bytes from the dex header file, update the checksum, checksum, fileSize, and headerSize fields to get the correct dex file. The specific process is shown in Figure 3.

4. Conclusion
In the field of mobile medicine, in order to improve the security of Android applications and prevent Android APP from malicious decompilation and tampering, a security reinforcement scheme for Android APP based on random confusion is proposed. Through a large number of random character confusion and dynamic loading of DEX header files, without affecting the execution efficiency of Android software, it increases the difficulty of decompiling DEX files, improves the security of source code, and ensures the correctness and integrity of DEX files.

References
[1] Misra A, Dubey A. Android Security: Attacks and Defenses[M]. Auerbach Publications, 2013.
[2] Chell D, Erasmus T, Colley S, et al. 6. Analyzing Android Applications[M]// The Mobile Application Hacker's Handbook. John Wiley & Sons, Inc. 2015:173-246.
[3] Zhi-Wen W U, Wei L I. Design and implementation of software reinforcement scheme based on Android platform[J]. Telecom Engineering Technics & Standardization, 2015.
[4] Zhao Y, Liu J. Analysis and Design of Android APP Security Reinforcement System[J]. Computer Engineering, 2018.
[5] Kovacs T, Egginton R. On the analysis and design of software for reinforcement learning, with a survey of existing systems[J]. Machine Learning, 2011, 84(1-2):7-49.
[6] Crussell J, Gibler C, Chen H. Attack of the Clones: Detecting Cloned Applications on Android Markets[J]. 2012, 81(13):2454-2456.
[7] Jung J H, Ju Y K, Lee H C, et al. Repackaging Attack on Android Banking Applications and Its Countermeasures[J]. Wireless Personal Communications, 2013, 73(4):1421-1437.
[8] Tsai K Y. Android App copy protection mechanism with semi-trusted loader[C]// International Conference on Advanced Communication Technology. Global IT Research Institute (GiRI), 2015:464-467.
[9] Falsina L, Fratantonio Y, Zanero S, et al. Grab 'n Run: Secure and Practical Dynamic Code Loading for Android Applications[C]// Computer Security Applications Conference. ACM, 2015:201-210.
[10] Yerima S Y, Sezer S, Mcwilliams G. Analysis of Bayesian classification-based approaches for Android malware detection[J]. Information Security Iet, 2014, 8(1):25-36.