Journal of Engineering and Applied Sciences 13 (Special Issue 3): 3261-3266, 2018
ISSN: 1816-949X

Improvement of Android Banking Application Integrity Using Dynamic Key Value

Ji-ho Cho, Chung-hyun Lim, Hyo-jung Ahn and Geuk Lee
Department of Computer Engineering, Hannam University, Dae-Jeon, Republic of Korea

Abstract: As the number of smartphone user increase rapidly over the recent several years, numerous android applications are developed. In the android market, due to the openness of the market, the distribution of not only normal applications but also, malicious applications disguised as normal applications are increasing. Malicious financial applications disguise themselves as normal applications to steal personal information, account passwords and security card information and attempt illegal transactions. Malicious banking applications easily evade integrity verification and are transformed into another malicious applications using repackaging method. Banking applications require very rigid integrity verification method. In this study, integrity evasion methods of android banking application are introduced and a verification method using dynamic key values to prevent the integrity evasion attack is proposed. Application program's memory load address is used as dynamic key value.

Key words: Android banking App., integrity verification, dynamic key value, evasion methods, banking application, dynamic key value

INTRODUCTION

The number of mobile banking customers in South Korea is 71.92 million as of March, 2016 which was smaller by 6.1% (-4.64 million) compared to the end of the previous quarter (76.56 million). This is attributable to the fact that the number of registered customers decreased drastically compared to the end of the previous quarter (-7.85 million). Since, the banking service based on IC chip and based on VM that had been used in feature phones from the beginning of mobile banking service was terminated at the end of 2015. Meanwhile, the number of registered customers who were using smartphone based mobile banking service was 68 million. It is increased by 5.0% (+3.21 million) compared to the end of the previous quarter and has been maintaining a steady increasing trend (Jung-Hyuk, 2015).

Normal Apps. (applications) can be transformed into similar malicious Apps. using a technology called repackaging. Repackaging is a process through which Apps. are inverse transformed into their initial forms. New Apps. are made by revising the initial form and inserting some new source codes into the initial form. Anyone who has some knowledge and understanding of Java language can manipulate App. easily using repackaging technology. After repackaging was found for the first time in 2011, the distribution of repackaging Apps. rapidly increased in diverse paths such as Google Android market, third party markets and P2P. A problem of repackaging Apps. is that these Apps. disguise user as it is normal Apps. to steal various kinds of information related to individuals. Therefore, the safety verification function of banking Apps. is important (Seon-Mi, 2012).

MATERIALS AND METHODS

Analysis tools and techniques

Analysis procedure: The analysis procedure to analyze the vulnerabilities of applications program in the android operating system is as shown in Fig. 1. APK files are unpacked using the apk tool to create a native library. This library is disassembled using android NDK's arm-linux-androideabi-objdump to analyze the assembly code. In addition, character strings can be analyzed using Linux's string commands. Modified APKs can be made by inserting debug logs into the smali codes and repackaging the smali codes using the apktool. These modified codes can be installed and executed in android devices to conduct execution log analysis and memory dump file analysis. The classes.dex files created after decompressing APK files are converted into JAR (Java archive) format. After the conversion, the files are converted into Java source codes using the sublime text 2 decompile tool to analyze the source codes (Kim *et al*., 2013).

Corresponding Author: Geuk Lee, Department of Computer Engineering, Hannam University, Dae-Jeon, Republic of Korea

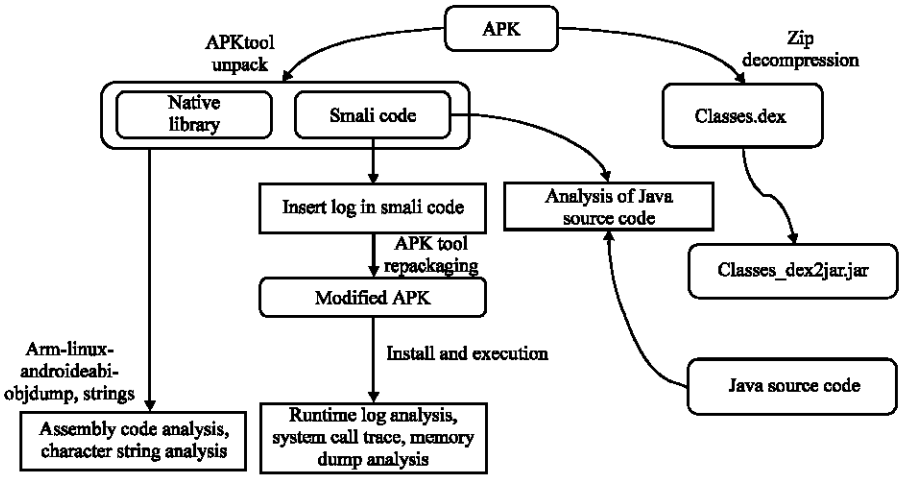

Fig. 1: Analysis procedure of android application

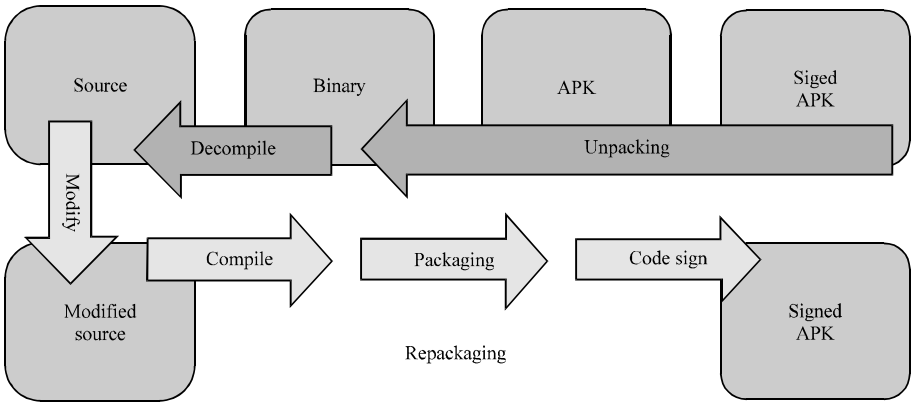

Fig. 2: APK unpacking and repackaging process

APK file unpacking and repackaging: Android Apps. can be effectively used by unpacking APK files, analyzing or revising the contents and repackaging the APK files. APK files are unpacked and repackaged using the apktool which is an android APK file reverse engineering tool. Apps. can be forged or falsified in this process.

When APK files are unpacked using the apktool, files and directories are extracted and a smali directory containing files created by disassembling android dex format is additionally created. The extracted directories and files are repackaged using the apktool and can be made into APK files that can be installed into the android operating system by signing using the jarsigner tool included in the Java SDK. Figure 2 shows the APK unpacking and repackaging process (Seon-Mi, 2012).

Java class decompile and source code analysis: APK files are in a ZIP compressed file format based on JAR (Java ARchive) file format. Therefore, the classes.dex files of APK files can be obtained by changing the extension name of APK files into ZIP and decompressing the files. classes.dex files can be converted into JAR files with simple command and the dex 2 jar program. Since, JAR files include Java class files, JAR files can be decompiled using Java decompiling tools such as sublime text 2. JAR files can be compiled again after conducting static analysis or revising the source codes on the source code level (Anonymous, 2010).

Where the parameter α is called embedding intensity and their effect of validity of the algorithm directly is apply after this process, after that apply the inverse wavelet transform to the image for find out watermark image (Fig. 3).

Memory dump analysis: In the android operating system, the heap memory of the executing App. can be dumped. To create memory heap dump, first an android studio is installed and an android emulator or mobile phone is

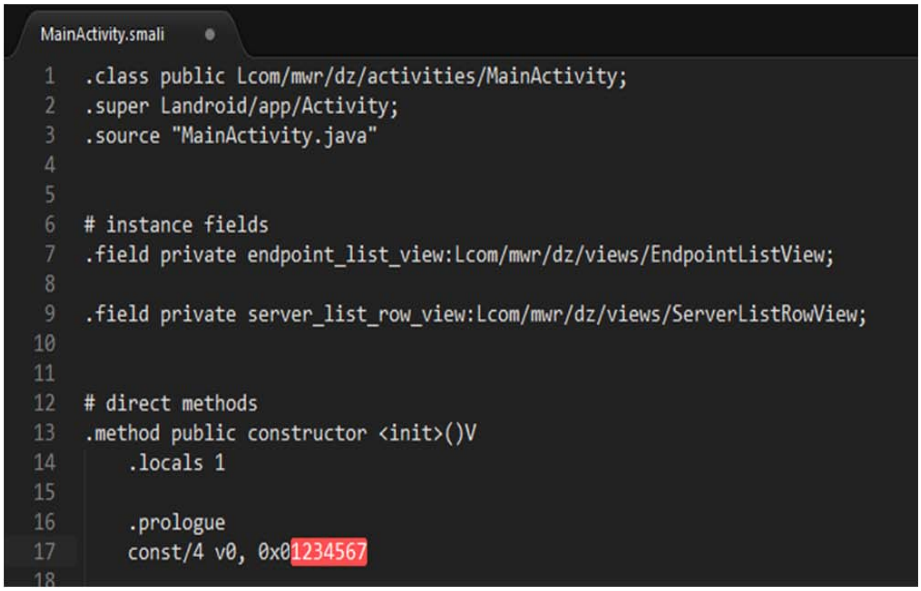

Fig. 3: Decompiled Java class using sublime text 2

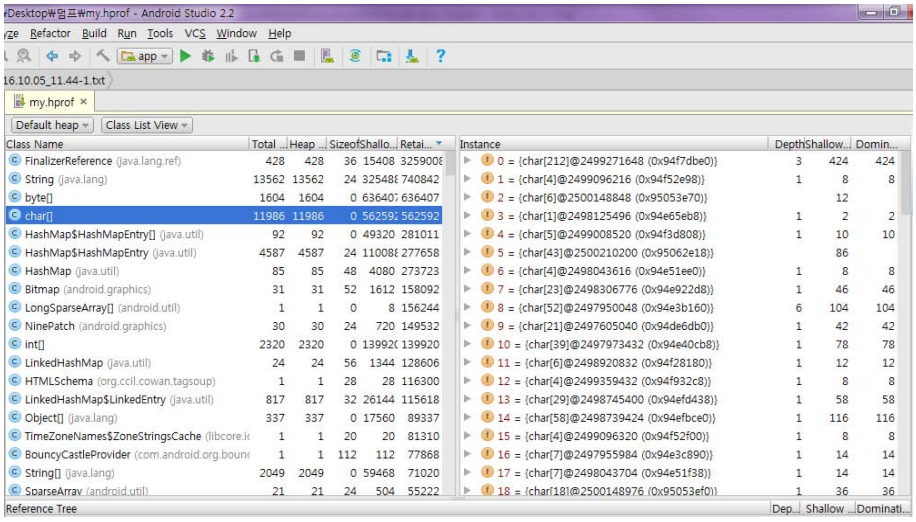

Fig. 4: Classes and instances of dump files

connected to the android studio. Next, the menu named android monitor in the android studio is executed. The memory usage of the executing App. can be seen by connecting the android studio and the App. player. In addition, dump files are created by pressing memory snapshot and dump file buttons in the android monitor. Figure 4 shows a snapshot and dump files (Anonmymous, 2015).

RESULTS AND DISCUSSION

Banking App. integrity evasion: When normally distributed android Apps. are forged and repackaged, the contents of forged APK files is changed to be different from the original ones. To detect this change, the hash value of the APK file of the executing App. in the smartphone is calculated, encrypted and transmitted to the server. The server compares the received hash value and the hash value of the original APK file stored in the server to verify the integrity of the App.

Every time an App. is executed, android loads the Java code included in the APK file into the Dalvik virtual machine. The path of the loaded APK file can be obtained from the App. being executed through the getPackageCodePath() or getApplicationInfo() method of the android.content.Context class.

Table 1: APK path evasion

Normal banking App.	APK path evasion method
The APK path is obtained from the Java code and transferred as arguments of the native function	The path value is changed in the Java code before it is transferred as arguments
Call getPackageCodePath() method from the native code	Change the return value by method overriding of the getPackageCodePath()
Call getApplicationInfo() method of from the native code	Change the return value by method overriding of the getApplicationInfo()

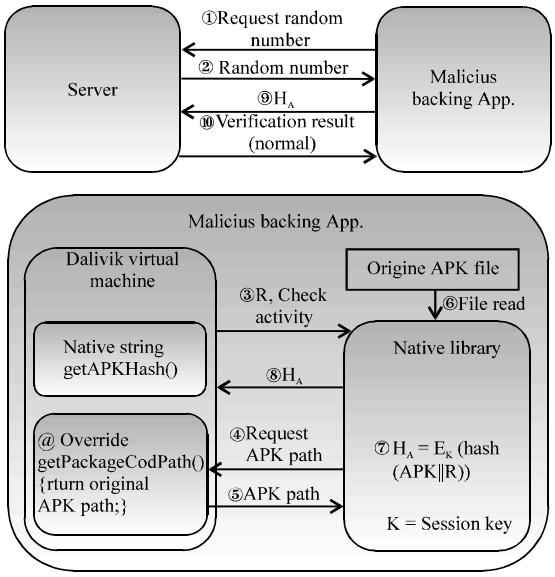

Fig. 5: Evasion of integrity verification

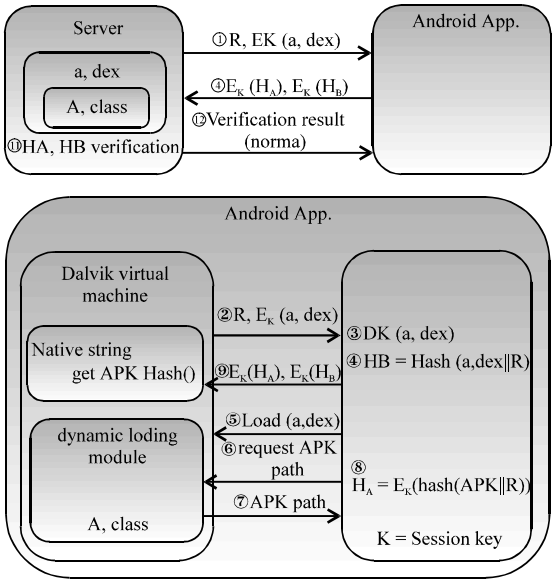

Fig. 6: Dynamic loading of Java code

To evade the integrity verification, forged App. need PAK path of original APK file. The forged App. includes the normally distributed original APK file in the forged App. Through smali code modification and repackaging, the forged APK file path used in the integrity verification is to be the APK file path of included original one. The original APK file is added to the assets directory of the forged App. The forged App. copies the original APK file in its data directory when it is executed (Fig. 5).

If the original banking App. used a method that obtains the path of the APK file from the Java code and transfer the path as arguments when call the native library function, the value can be changed very easily in the Java code of the forged App. Summary and comparison with normal App. of this attack is in Table 1 (Kim *et al*., 2013).

Prevention methods

Dynamic loading of Java code: To cope with the banking App. vulnerability of integrity verification explained in previous chapter, dynamic loading method of Java code is proposed (Kim *et al*., 2013). The process is shown in Fig. 6.

First, the file named a.dex necessary for integrity verification is stored in the server. This file is encrypted using session key K. The file is transmitted to the android App. together with the random number R. The session key K value is a symmetric key value for encryption and decryption of transmitted/received data. This value is exchanged by the banking App. and the server when the user session begins. All existing banking Apps. have session key K because they provide the encryption of transmitted/received data. After decrypting the a.dex using session key K, the android App. calculate the hash values of the a.dex and R value to obtain the integrity verification value H_B. The a.dex is loaded into the Dalvik virtual machine. The APK path is requested by the dynamic loading module Table 2. Next, apply the hash on the APK path and the random number in order to create integrity verification value H_A. In the final stage, integrity verification hash values H_A and H_B are separately encrypted using session key K to make $E_K(H_A)$ and $E_K(H_B)$ and these values are transmitted to the server. The server verifies the integrity of the banking App. and the a.dex file with the dynamic loading. Detail of this method is explained in the reference number 4.

In the Java code dynamic loading method, the server transmits the a.dex file to the App. An a.dex file is usually 1~4 MB in size. Whenever the user just press the App. execution button, an amount of data of 1~4 MB is used. In addition, the APK path is requested in step in Fig. 6.

Table 2: Strengths of the Java code dynamic load technique

Problem of banking Apps.	Solution of Java code dynamic loading	Strength
Vulnerabilities in verification using the APK path	A.dex Java code is dynamically loaded	Method overriding vulnerabilities are resolved

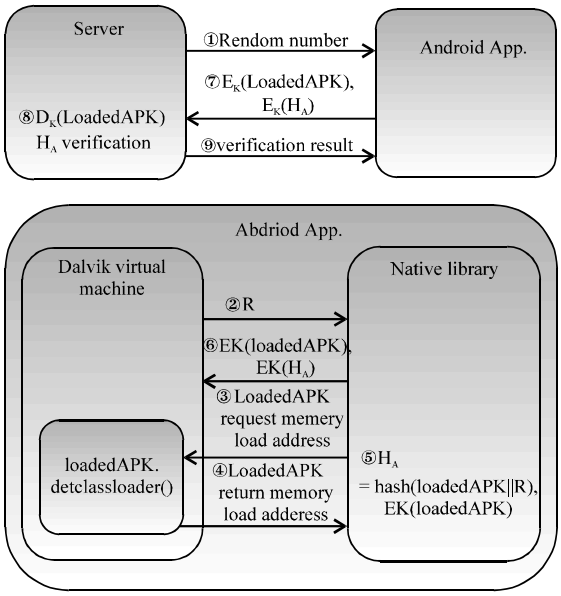

Fig. 7: Integrity verification method using dynamic key value

Since, the APK path once installed is invariably stored at the same position, APK path could be known to the hacker in several ways.

Integrity verification using dynamic key value

Dynamic key value technique: When an application is executed, the loading address of the program is changed whenever it is loaded in the main memory. We proposes a method that uses the memory address as a key value that is changed in dynamic.

The integrity verification method using the dynamic memory load address when an application is executed is as shown in Fig. 7. First, the android App. requests the server for the random number R and receives it. This random number R value is sent to the library. Next, the Android App. executes the LoadedAPK.getClassLoader() function to receive the memory address of LoadedAPK class as a key value. The native library calculates H_A value using the hash values of the LoadedAPK and the random number R. The LoadedAPK value and the H_A value are separately encrypted into session key K values to transmit $E_K(H_A)$ and E_K(LoadedAPK) to the server. The server decrypts $E_K(H_A)$, E_K(LoadedAPK) received from the App. The decrypted D_K(LoadedAPK) and random number R value are hashed. The verification result is send to the android App. to complete the verification. Figure 7 shows this process.

The evasion attack can be prevented using the method shown in Fig. 7. In addition, the method consumes less data space than the method using the loading module of Java. It also consumes less time because it need not encryption and decryption process of the a.dex file.

Table 3: Weaknesses of the Java code dynamic load technique

Category	Weakness
a.dex file	1~4 MB of data is consumed due to a.dex file transmission when the App. is executed a.dex should be encrypted and decrypted
APK path	The path is not changed once the APK file is stored

Table 4: Improvements of the proposed method

Dynamic loading of code	Proposed method	Improvements
Transmit the Java code to prevent the vulnerability of method override	Use a load address as the key value	No a.dex file encryption and decryption process Less data and time are consumed No dynamic loading module is used

Comparison with dynamic loading of code: Whereas, the techniques use dynamic loading modules to make the Java code to be dynamically executed, the technique proposed in the study does not use any dynamic loading module but does use the dynamic key value for verification (Table 3).

When a dynamic loading module is used, the a.dex file should be received from the server and should be decrypted. And the random number R and hash value of a.dex are used to calculate H and H is encrypted. However, since, the proposed method need not use dynamic loading module, the processes to encrypt and decrypt a.dex file were skipped. Dynamic code loading method needs some data communication since the a.dex file should be received every time the application is executed to enable the verification. The proposed method enables verification without the receiving/sending the a.dex file making verification fast and simple (Table 4).

CONCLUSION

In this study, first, we analyze vulnerabilities of the android banking App. integrity verification function. Second, we introduce a dynamic loading method of Java code to prevent the evasion attack of integrity verification. Third, dynamic key value method is proposed. The dynamic key method uses the memory address that can be obtained in loadAPK class as the key value. This value is loaded address of application program and it varies whenever it loaded.

In the dynamic loading method of Java code, the a.dex file is loaded dynamically. In this process, the a.dex file is received from the server, encrypted and decrypted. The method proposed In the study does not uses the dynamic loading module but uses the dynamic key value. Therefore, less data space is needed and the a.dex file send/receive with the server is not need and the processes to encrypt and decrypt the a.dex file was also skipped. The proposed dynamic key value method is more convenient and simple than that of dynamic loading module method.

RECOMMENDATIONS

More studies are needed in integrity assurance areas to improve integrity verification of application programs and to cope with the new attacking technologies on the integrity verification.

ACKNOWLEDGEMENT

This research was supported by 2017 Hannam University Research Fund.

REFERENCES

Anonmymous, 2015. [How to use NEXT android profiler]. LinkedIn Corporation, Sunnyvale, California, USA. (In Korean) https://www.slideshare.net/arload/next-android-profiler.

Anonymous, 2016. Android (operating system). https://ko.wikipedia.org/wiki/Android(operating_system),_%E2%80%9Candroid(operating_system)%E2%80%9D.

Jung-Hyuk, K., 2015. The present situation of use of domestic Internet banking services in the third quarter of 2015. Bank of Korea, Washington, DC., USA.

Kim, S., S. Kim and D.H. Lee, 2013. A study on the vulnerability of integrity verification functions of android-based smartphone banking applications. J. Korea Inst. Inf. Secur. Cryptology, 23: 743-755.

Seon-Mi, G., 2012. Thieves that secretely stealed into smartphone. AhnLab, Inc., Gyeonggi Province, South Korea. http://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?seq=19986.