A Method for Detecting Illegally Copied Apk Files on the Network

Sungmin Kim
Soongsil University,
1-1 Sangdo-Dong, Dongjak-gu,
Seoul 156-743, Korea
+82-2-824-3862
smkim@ssu.ac.kr

Eunhoe Kim
Soongsil University
1-1 Sangdo-Dong, Dongjak-gu,
Seoul 156-743, Korea
+82-2-824-3862
ehkimnet@ssu.ac.kr

Jaeyoung Choi
Soongsil University
1-1 Sangdo-Dong, Dongjak-gu,
Seoul 156-743, Korea
+82-2-820-0684
choi@ssu.ac.kr

ABSTRACT
We present a method of detecting illegally copied Android apps targeting at the APK files. We extracted the data objects being transmitted from the network through the process of sniffing, analyzing, and assembling packets. Then, we made an analysis on the features of APK files to judge whether the extracted data object is an APK file or not. We were able to achieve a success rate of 95.5% in detecting illegally copied apps by identifying them through APK feature points and forensic technologies.

Categories and Subject Descriptors
D.2.9 [**Software**]: Management – *copyrights*. C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks - General – *Security and protection*.

General Terms
Security, Management, Design

Keywords
Android app, illegally copied app, forensic watermarking, APK feature point, packet sniffing

1. INTRODUCTION
Along the Gartner survey[1] for worldwide smartphone slaes,Android application markets reached more than 52.5% in smart mobile market. In this situation, the use of illegally copied apps is also increasing very rapidly. In case of Angry Bird, a popular Android app game, its replication rate reaches almost 97%. This case suggests that protecting app developers' copyrights is weakening. In the long term, it may lead to a slump in Android Application market and a serious problem that loses transparency in its official trading, due to the distribution and use of illegally copied applications.

The Android operating system basically protects its inner system memory area against the reckless access from the outside, so it prevents app from being copied and maintains the transparency of its inner system. However, users can use Rooting [2], a method for granting the authority of super users. Therefore, the current Android operating system is available for illegally copied apps, so it is needed to make a method for preventing the distribution of illegally copied app files.

In this paper we propose a method for detecting illegally copied apps on the network, where all files are being transmitted, in order to prevent the distribution of illegally copied apps. The proposed method aims at HTTP and FTP protocols, which are most widely used for transmitting illegally copied APK files on webhards and web servers, which are main distributors of illegally copied apps. We use an identification technology based on the APK feature points and the signature-based forensic watermarking. In addition, we propose a method for sniffing packets at the network device driver level and storing them in a repository. The stored packets are analyzed, compounded, and identified later if necessary.

RACS'12, October 23-26, 2012, San Antonio, TX, USA.

2. RELATED WORKS
There are several works to detect illegally copied apps and prevent them from executing or installing [3][4]. Most of them use forensic watermarking methods based on signature technology [5]. The systems for blocking illegal copied apps [3][4] can extract forensic watermarking information through a system software which had already been set up at the Android platform. After extracting forensic watermarking information, these systems can detect illegally copied apps, which can be prevented from executing or installing. They could show good performance for detecting illegally copied apps because they use forensic watermarking methods [5]. However they are not compatible with various Android OS versions.

There is another study that uses an online technology for controlling execution of illegally copied apps [6]. In the online technology, a part of execution code places on a separate remote server. Therefore, a user has to download the part of the execution code for executing an app. To block executing illegally copied apps, the server authenticates users shall apply their purchase information. But this online technology is affected by networking environments and it needs extra cost of data communication for downloading the codes from the server.

3. DETECTOR OF TRANSMISSION OF ILLEGAL ANDROID APPS
A detector of illegally copied Android apps during their transmission consists of 4 modules – packet sniffing module, packet analysis module, packet combination module, and finger printing extraction module, as shown in Figure 1.

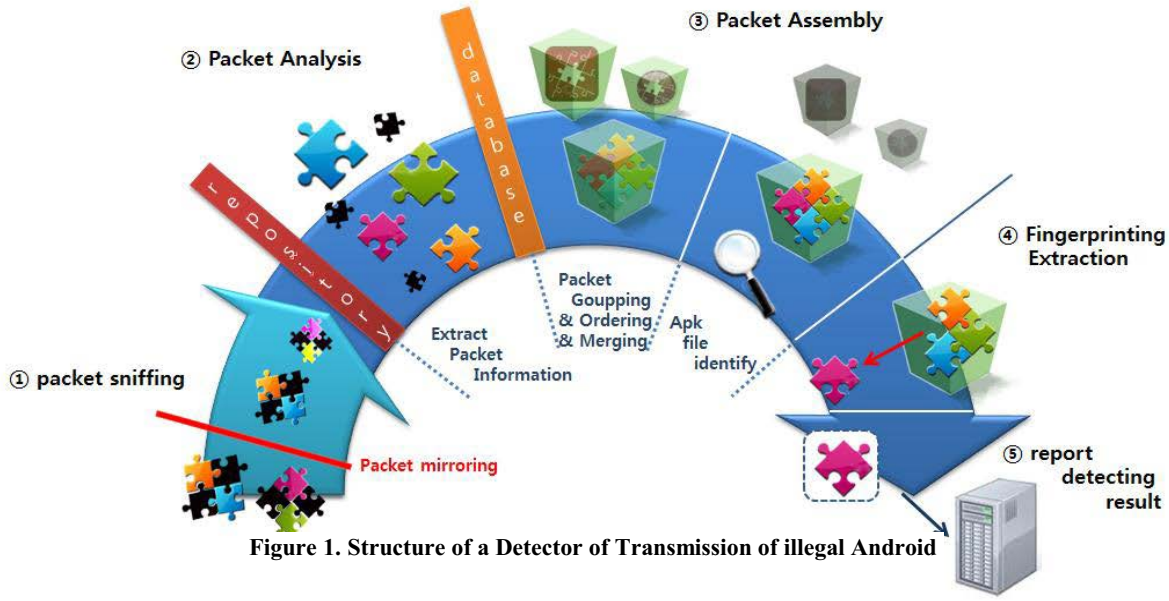

Figure 1. Structure of a Detector of Transmission of illegal Android

3.1 Packet Sniffing Module

Packet sniffing module copies all kinds of packets delivered through the network and stores them in the repository in real-time. To increase a detection rate of illegally copied apps, the packet module is located in the network gateway where packets are clustered. In addition, to minimize the slowdown of the network gateway, packet sniffing is performed at the network device driver level in TCP/IP layer 1 as shown in Figure 2.

In the Linux kernel, the network device driver is provided as kernel shared library type unlike a block device or a character device. For this, a network device driver consists of 3 modules - network driver layer, network devices interface layer, and network devices layer as in Figure 2. The network devices interface layer offers the abstract interface to connect between a network device layer and a network driver layer.

The network device layer is independent of the adapter. To increase the compatibility with various network device drivers, the packet sniffing module is installed to the network device layer of the layer 1 of TCP/IP stack. Therefore, it minimizes the system degradation on packet sniffing and guarantees independency of the adapters.

3.2 Packet Analysis Module

A packet analyzing module is applied to analyze the packet data obtained in the sniffing process. It extracts and stores the packet data using only HTTP and FTP protocols in the database. Figures 3 and 4 show the field information of IP and TCP header actually used for packet analysis, respectively.

The processes of packet analysis are as follows: at first, this packet analysis module refers to the version information of IP head and excludes the packets not corresponding to IP version 4. Secondly, it refers to the IP head length value (IHL), and calculates the TCP head's starting location. Then, for packet assembly, it stores the information on the identification number and the fragmentation offset in database. Lastly, it refers to the protocol field value and excludes packets that are not corresponding to 0X06 (TCP). It is because a detector of illegally copied apps considers only the HTTP and FTP protocols. Finally, source IP address and destination IP address are stored in database for packet assembly.

A packet analyzing module moves to the TCP head starting point after completing the analysis on IP head information. Firstly, it extracts the source port value and the destination port value needed in packet assembly. Secondly, it stores a sequence number and an acknowledgement number in database. Lastly, the packet analyzing module refers to data offset values and moves to the data field starting point. If HTTP unique strings (48 54 54 50) do not exist in the data field or the source port value and the destination port value are neither HTTP (80) nor FTP (20, 21), packet analysis module excludes them all.

3.3 Packet Assembly Module

Each packet data, transferred using HTTP or FTP and stored in database, is not enough to identity whether it is a part of app file or not. A packet assembly module merges each related packet using the packet head information stored in the database and extracts objects being transmitted from the database. Then, this module identifies whether the extracted object is an APK file or not. If this object is identified as an APK file, it stores the file as a temporary file. The packet assembly module consists of packet grouping, packet ordering, and an APK file identification process.

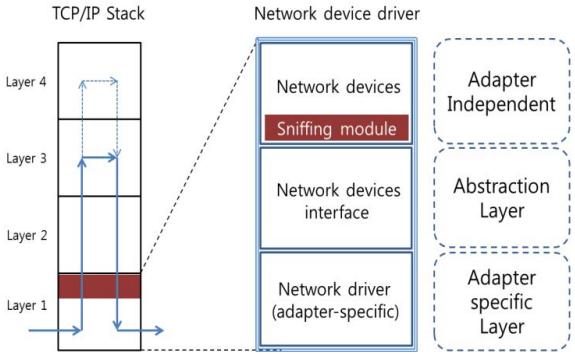

Figure 2. Location of the Sniffing Module in TCP/IP Layer

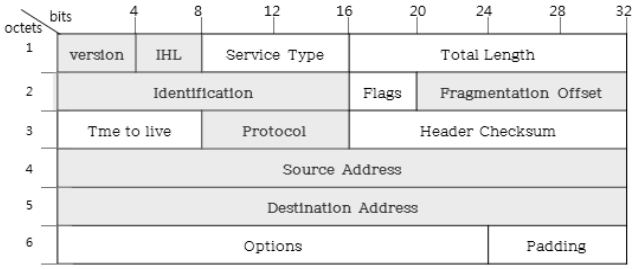

Figure 3. IP Header Extraction Factors

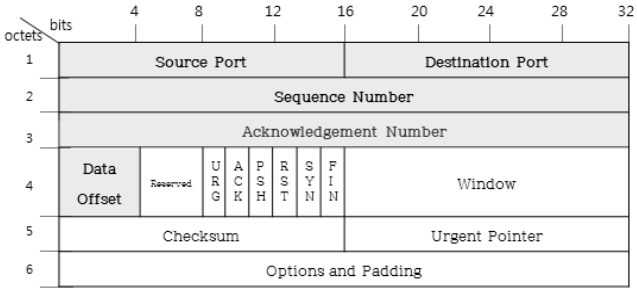

Figure 4. TCP Header Extraction Factors

3.3.1 Packet Grouping and Ordering

In order to reconstruct objects from transmitting packets, first of all, it needs to arrange packet data into each related group. When the data is transmitted using HTTP and FTP, it is necessary to set up communication connection in the network and the data divided up into packet units is transmitted. Therefore, to reconstruct the original transmitted data from packets, it is needed to make a packet group by the communication connection. For this, a packet assembly module combines source IP address, destination IP address, source port, and destination port values. These elements make the unique keys to identify data transfer session.

To reconstruct the transmitted data, it is needed to line up grouped packets following the original order. Therefore, the packet assembly module is necessary to line up the packets through packet ordering. Generally, when the data is transmitted, this data is divided into many datagrams in IP layer. First, the module refers to the identification number of the IP head to adjust the order of divided data in IP layer, because the unique identification number in the IP layer is randomly generated and it is allocated in the first packet. This identification number has increased as much as 1 in a series of following packets. Depending on the network environment, the sizes of data transmitted at a time are different each other. Therefore, when one IP packet is transmitted, it may be divided into several IP packets depending on the network environment. The divided IP packets use the same identification number and order by fragment offset values. Thus, the packet ordering process is referencing with both IP layer identification and fragment offset values. Next, it refers to the sequence numbers of TCP head for adjusting the order of packets, and lastly checks retransmitted packets by reference to the acknowledge numbers. After arranging packets in each groups, the packet assembly module assembles each objects corresponding to the data transfer sessions.

3.3.2 APK Files Identification

The data type transmitted using HTTP and FTP protocols are various, so it cannot guarantee the data objects extracted in the process of grouping and ordering packets are all of an APK file. Therefore, before identifying whether a certain APK file is an illegal file or not, it is necessary to determine whether extracted objects are an APK file or not. For this, we find out features of APK files and apply them. The first feature of an APK file is that all the APK files have a zip file structure. The second one is that all APK files have a promised string to inner head of an APK file regardless of file size.

3.3.2.1 Checking a Zip File Structure

An APK file, an Android app, has the structure of a jar file. And the jar file uses the compression method identical to a zip file. Therefore it is possible to determine the Android apps by checking whether extracted objects have the structure identical to a zip file. As in Figure 5, the zip file consists of the local header area, the central header area, and the end of record area. The first area, local header, contains individual specific information of compressed files. The starting point of each compressed file in local header area is marked using unique codes (50 4B 03 04). And each compressed file contains specific information such as the compressed file name, compressed rates, file size, and compressed data. The second area, central header, starts with unique codes (50 4B 05 06) and stores the information of all recorded local headers. Lastly, the end of central recorder area starting with unique codes (50 4B 01 02) has the overall structure information of the zip file, such as the length value of the local header region (starting point of the central header), the sizes of all files, CRC codes, etc. In a zip file structure, each unique codes are located on the starting points of the local header, the central header, and the end of record header. These unique codes are the unique features of zip files. Therefore, the APK files using the zip file structure have these unique codes. So, it is needed to check whether these unique codes exist or not for judging only APK objects.

3.3.2.2 Checking Apk File Feature Points

APK files are created by using apkbuilder, jarsigner, and zipalign provided in Google [7]. Apkbuilder is a tool for making APK files. It bundles classes.dex, AndroidManifest.xml, resources.arsc, resource files, and external files into an APK package.

Classes.dex file is an integrated compiled executable file so that the app execution codes formed in Java language can be executed in a Dalvik machine. AndroidManifest.xml file includes the

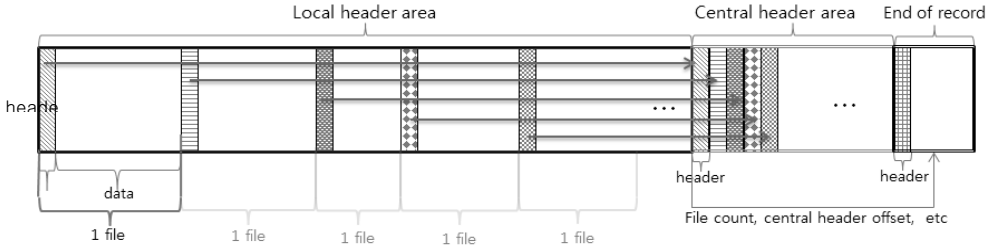

Figure 4. Zip file Structure

information and authorities related to apps installation and the set-up information. This file is compressed and included in an APK package. Resources.arsc file has the information of resource files used in developing apps. Resources files are used for apps' inner set-up, such as the images, sound sources, and layouts. Those are located in folders taking res directory as their parent. The assets directory for fixed external files is not compressed but included in the APK package. However, this directory doesn't use a fixed folder structure in it.

Apkbuilder packages these files, and Jarsigner prepares the distribution of apps by attaching their unique signatures to the created package. Lastly, zipalign compresses this package in a zip file format and creates a final APK file. All kinds of apps have the above files in common and the directories included in the process creates the APK package through Android app development tools. Thus, an APK has the promised file names which the APK file have to include, so they are recorded in local headers in the process of zip format compression. For this reason, if each string of classes.dex, AndroidManifest.xml, resources.arsc, res/, and assets/ exists in local header, this object can be an APK file. Consequently, the objects having zip file structure and including above file names are stored as final object files.

3.4 Forensic Watermarking Information Extraction

In this paper, there is a precondition to prevent illegally copied apps. Forensic watermarking information has to be inserted in a dex file, classified as a Java executable code file. Forensic watermarking information basically includes the purchaser information of an APK file. The dex file is packaged in an APK file, so all the APK files transmitted from official app markets have dex files where forensic watermarking information is inserted into. Therefore, forensic watermarking information extraction is attempted in the dex files of the temporary app files identified as an APK file by a packet assembly module. When the forensic watermarking information is extracted, the APK file investigates whether source IP address of the APK file is one of IP addresses belonged to official app markets. If the source IP address is not one of official market IP addresses, the apk file is determined as an illegal app.

3.5 Report Detecting Result

After extracting forensic watermarking information and detecting illegally copied apps, the information about illegally copied apps are sent to a central reporting server. The information includes illegal app information, forensic watermarking information, downloader ip address, source ip address, download request time, and protocol information. This information is managed on a control module of the central server, which provides reporting web pages

4. TEST RESULTS AND ANALYSIS

The test was conducted in the environment of Inter(R) Core(TM)2 Duo CPU E8400 (3.00GHz, 4GB)/2.6.32-33 as Linux kennel version. The Intel(R) Pro/1000 PT DUAL PORT 20GB was used as a network card. An e1000e driver was applied as a network device driver. We installed a router in the prepared PC and mounted a detector of illegal app transmission.

Table 1. Test Results

Test Result Items	Top
Total number of sniffed - packets	225420
Number of assembled apk files	385 (96.2%)
Number of extracted water markings	382 (95.5%)

For this test, we prepared various APK files with forensic watermarking information. And these APK files were randomly downloaded 400 times for 1 hour. We used a test client program applying FTP and HTTP protocols. A result of test is Table 1.

5. CONCLUSION

We proposed a method to detect illegally copied app distribution in the network. The proposed detector for illegally copied apps has advantages of minimizing the slowdown of operating systems due to illegally copied app detection, as it performs packet sniffing on the network driver level. In addition, we proposed a method for identifying the illegality of apps by judging the apps based on apk feature points and forensic watermarking information. Thus we could obtain 95.5% of a high detecting rate. We are also planning to analyze packets and study assembly process in order to apply them to various protocols. Moreover, we will sophisticate the identification technologies to increase a success rate of forensic watermarking extraction afterward.

6. ACKNOWLEDGMENTS

This research project was supported by Ministry of Culture, Sports and Tourism (MCST) and from Korea Copyright Commission in 2012.

7. REFERENCES

[1] Gartner press releases report of Mobile Market Share Devices by Region and Country 3Q11, 2011.

[2] Woo Bong Cheon, *A Study on Jamming Vulnerability of Aeronautical Communication System Using Android Phone.* Information Science and Applications International Conference, 2011.

[3] Eungyu Lee, Yeongung Park, Kanghee Kim, Seongje Cho, *A Mobile Application Anti-Piracy Technique Using Mandatory Access Control*. KIISSE Dependatble Computing, 2012.

[4] Sung-Ryul Kim, Copy Protection Systemfor Android App using Public Key Infrastructure. Security Engineering Research Support Center, 2012.

[5] Joonhyouk Jang, Cheol Jeon, Bongjae Kim, Jiyeon Park, Wookun Cho, *Robust Static Software Watermarking Schem for Copyright Protection of Mobile Software.* KIISSE Dependatble Computing, 2012.

[6] H.W.Jung, J.S.Lee, Y.H.Suh, Reaserch Trend of Illegal Contents Trace Technology, Electronics and Telecommunications Reaserch Institute. 20, 2005, 120-128.

[7] Google Android developer reference paper, http://developer.android.com/guide/developing